

# Чекпоинт 4, команда 67

## Предсказание динамики физической системы с помощью нейросетей (годовой проект)

### Оглавление

0. Введение.....	2
1. Серверная часть (FastAPI) .....	2
1.1. Структура программы FastAPI.....	2
1.2. Настройки программы и процесса обучения моделей.....	2
1.3. Схема данных .....	4
1.4. Доступные запросы .....	5
2. Клиентская часть (Streamlit) .....	7
2.1. Структура программы Streamlit .....	7
2.2. Настройки программы .....	7
2.3. Функции запросов к серверу (client_funcs.py) .....	8
2.4. Интерфейс стартовой страницы приложения.....	9
2.5. Интерфейс страницы просмотра разведочного анализа данных.....	9
2.6. Интерфейс страницы обучения и инференса моделей.....	11
2.6.1. Кнопка «Обучить модель» .....	12
2.6.2. Кнопка «Сделать предсказание» .....	15
2.6.3. Кнопка «Список моделей».....	16
3. Развёртывание FastAPI и Streamlit.....	19
3.1. Локальное развертывание.....	19
3.2. Развёртывание в Docker .....	19
3.2. Развёртывание на VPS .....	19

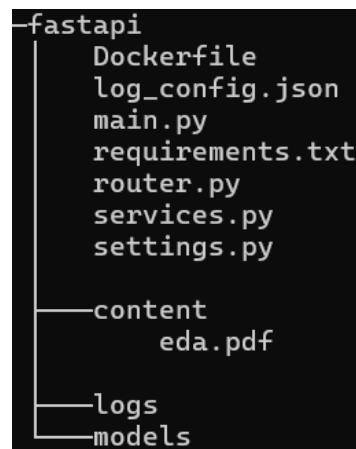
## 0. Введение

Весь код к этому чекпоинту реализован в папке **service**. Папка **telegram** к чекпоинту не относится, телеграм-бот будет реализован в перспективе.

### 1. Серверная часть (FastAPI)

#### 1.1. Структура программы FastAPI

Сервер реализует функции обучения и инференса моделей, решающих задачу классификации событий эксперимента LHCb (детектор на Большом адронном коллайдере). Структура программы FastAPI представлена на рисунке:



Наличие некоторых файлов и папок требует пояснения:

- **main.py** – точка входа программы;
- **router.py** – файл API-роутера, в котором определены функции обработки запросов;
- **settings.py** – файл, в котором определена конфигурация программы;
- **services.py** – файл, содержащий функции для обучения моделей;
- **log\_config.json** – файл конфигурации логирования для сервера uvicorn;
- **content** – папка, содержащая готовые файлы для отправки клиенту (в частности, файл с выполненным разведочным анализом данных **eda.pdf**);
- **logs** – папка с логами;
- **models** – папка с обученными моделями.

Папки **logs** и **models** создаются автоматически, если на момент запуска сервера их не существовало.

#### 1.2. Настройки программы и процесса обучения моделей

Конфигурация программы определена в классе Settings (`settings.py`), унаследованном от `pydantic_settings.BaseSettings`. Для программы определены следующие переменные конфигурации:

- **MODEL\_DIR: str** – путь до папки, в которую сохраняются обученные модели (по умолчанию это `models`);
- **LOG\_DIR: str** – путь до папки, в которую сохраняются обученные модели (по умолчанию это `logs`);
- **PDF\_PARH: str** – путь до PDF-файла с разведочным анализом данных;
- **NUM\_CPUS: int** – максимальное число доступных для обучения CPU (т.е. максимальное число процессов, в которых можно запустить обучение модели);
- **TARGET\_COL: str** – столбец, содержащий целевую переменную (в нашем случае это столбец `Label` с типом зафиксированного в ходе эксперимента LHCb события – фоновое событие или сигнал);
- **SIGNAL: str** – обозначение сигнального события в столбце, содержащем целевую переменную;
- **NON\_FEATURE\_COLS: List[str]** – столбцы, не используемые при обучении;
- **INDEX\_COL: str** – столбец индексов (в нашем случае это столбец `EventId` с номерами зафиксированных событий эксперимента LHCb);
- **TIME\_LIMIT: int** – предел времени обучения моделей в секундах;
- **MODEL\_TYPES: List[str]** – доступные для обучения типы моделей (пока доступны только логистическая регрессия, SVM, случайный лес и градиентный бустинг);
- **DATASET\_COLS: Dict[str, str]** – полный перечень столбцов датасета с указанием типа;
- **AVAILABLE\_SCORINGS: List[str]** – доступные метрики качества моделей (`accuracy` и `f1`);
- **LOG\_CONFIG\_PATH: str** – путь до файла конфигурации логирования сервера `uvicorn` (`log_config.json`; логи записываются в файл, реализована ротация логов по достижении размера файла около 0,5 МБ).

Кроме того, некоторые изменяемые в ходе выполнения программы настройки (относящихся к обучению и инференсу модели) реализованы в классе Services (`services.py`):

- **ACTIVE\_PROCESSES: int** – количество активных процессов обучения;
- **MODELS\_LIST: Dict[str, sklearn.pipeline.Pipeline]** – словарь с обученными моделями (содержит пары «ID модели – объект Pipeline»);
- **MODELS\_TYPES\_LIST: Dict[str, str]** – словарь с типами конкретных моделей (содержит пары «ID модели – тип модели»);

- **CURRENT\_MODEL\_ID: str** – ID текущей модели, которая установлена для инференса.

В том же классе Services реализованы методы для обучения и инференса моделей:

- **read\_existing\_models** – чтение моделей из папки MODEL\_DIR при запуске сервера;
- **fit(X: List[List[float]], y: List[float], config: Dict[str, Any]) -> Dict[str, Any]** – обучение модели с проверкой гиперпараметров и контролем времени обучения моделей. Метод возвращает словарь с ID модели и статусом ее обучения;
- **find\_id(model\_id: str) -> bool** – поиск модели в списке моделей;
- **predict(X: List[List[float]], model\_id: str) -> List[float]** – выполнение предсказаний при помощи модели с заданным ID;
- **compare\_models(X: List[List[float]], y: List[float], ids: List[str]) -> Dict[str, float]** – сравнение качества моделей. Метод возвращает словарь вида «ID модели – словарь “название метрики – значение”»;
- **remove(model\_id: str)** – удаление модели с заданным ID из списка моделей;
- **remove\_all(self) -> List[str]** – удаление всех обученных моделей. Метод возвращает список ID удаленных моделей;
- **get\_params(model\_type: str) -> List[str]** – получение списка гиперпараметров для модели заданного типа.

### 1.3. Схема данных

Рассмотрим классы запросов и ответов, которые используются при взаимодействии клиента с сервером (все они унаследованы от pydantic.BaseModel и определены в router.py):

- **MessageResponse(message: str)** – универсальный класс для ответа сервера на различные запросы, когда от сервера требуется текстовое сообщение;
- **ModelConfig(id: str, type: Literal["LogReg", "SVM", "RandomForest", "GradientBoosting"], hyperparameters: Optional[Dict[str, Any]])** – класс конфигурации моделей машинного обучения: id – ID обучаемой модели, type – тип модели (логистическая регрессия и т.д.), hyperparameters – перечень гиперпараметров в виде словаря;
- **DataColumnsResponse(columns: Dict[str, str], target: str, non\_feature: Optional[List[str]])** – ответ сервера, содержащий столбцы датасета: columns – все столбцы с указанием типа; target – целевая переменная; non\_feature – столбцы, не используемые при обучении;

- **ModelTypesResponse(models: Dict[str, List[str]])** – ответ сервера с перечнем типов моделей и их гиперпараметров: models – словарь доступных для обучения типов моделей (в виде пар «тип модели – список гиперпараметров»).

**Из списка гиперпараметров намеренно удален параметр n\_jobs!**

- **IdResponse(id: str, status: Literal["load", "unload", "trained", "not trained", "removed", "error"])** – ответ сервера с информацией о состоянии конкретной модели: id – ID модели, status – состояние модели (всего 6 состояний):

- а) модель установлена для инференса (load);
- б) модель снята с инференса (unload);
- в) модель обучена (trained);
- г) обучение модели прервано из-за превышения временного лимита (not trained);
- д) модель удалена (removed);
- е) ошибка обучения модели из-за неверных гиперпараметров и т.п. (error).

- **RequestError(detail: str)** – ответ сервера в случае ошибки (HTTPException): detail – сообщение с подробностями об ошибке;

- **PredictResponse(predictions: List[float], index: List[float], index\_name: str)** – ответ сервера с предсказаниями модели: predictions – столбец предсказаний, index – столбец индексов, index\_name – название столбца индексов (для формирования pandas.DataFrame на клиентской стороне);

- **CompareModelsRequest(ids: List[str])** – запрос на сравнение качества моделей по доступным метрикам: ids – список ID сравниваемых моделей);

- **CompareModelsResponse(results: Dict[str, Dict[str, float]])** – ответ сервера со сравнением качества моделей: results – результаты сравнения в виде словаря «ID модели – словарь “название метрики – значение”»;

- **ModelsListResponse(models: List[ModelConfig])** – ответ сервера со списком обученных моделей models.

#### **1.4. Доступные запросы**

Ниже перечислены доступные запросы на сервер:

1. **«Корневой» GET-запрос.** Реализован вне API-роутера. В качестве ответа сервер возвращает MessageResponse(message="Ready to work!")

Остальные запросы реализованы для API-роутера с префиксом /api/model\_service.

2. GET-запрос на получение PDF-файла **/get\_eda\_pdf**. Сервер возвращает FileResponse с заданным в настройках PDF-файлом.

3. GET-запрос списка столбцов датасета с их типами **/get\_columns**. Сервер возвращает `DataColumnsResponse`.

4. GET-запрос списка типов моделей, которые можно обучить, **/get\_model\_types**. Сервер возвращает `ModelTypesResponse`.

5. POST-запрос обучения модели **/train\_with\_file**. Входные данные: строка `models_str`, из которой формируется список конфигураций моделей `ModelConfig`, и файл `file` с тренировочным датасетом. Обучение происходит для каждой модели асинхронно в отдельном процессе: при этом, если количество моделей превышает количество доступных CPU, или ID моделей в `models_str` дублируются, или модель с каким-либо ID уже обучалась ранее, сервер возвращает `RequestError` без запуска обучения. В процессе обучения также могут возникнуть следующие ошибки:

- ошибки в гиперпараметрах и т.п.;
- превышение предельного времени обучения `TIME_LIMIT`.

В этом случае `RequestError` не возвращается, а модели с тем или иным ID присваивается статус в соответствии с `IdResponse`.

По завершении обучения сервер возвращает список `IdResponse`

6. GET-запрос на получение текущей модели **/get\_current\_model**. Если текущая модель для инференса установлена, сервер возвращает `MessageResponse` с ID; в противном случае сервер возвращает `RequestError`.

7. POST-запрос на установку модели с заданным ID для инференса **/set\_model/{model\_id}**. ID модели передается как Path-параметр. Если ID не найден или модель уже установлена для инференса, сервер возвращает `RequestError`. В противном случае – `IdResponse(id=model_id, status='load')`.

8. POST-запрос на снятие текущей модели с инференса **/unset\_model**. Модель для инференса не найдена, сервер возвращает `RequestError`. В противном случае – `IdResponse(id=model_id, status='unload')`.

9. POST-запрос обучения модели **/predict\_with\_file**. Входные данные – файл `file` с датасетом для предсказаний. Если никакая модель для инференса не установлена, сервер возвращает `RequestError`, иначе – `PredictResponse`.

10. POST-запрос на сравнение качества моделей **/compare\_models**. Входные данные: строка `models_str`, из которой формируется объект `CompareModelsRequest`, и файл `file` с датасетом для сравнения (датасет должен включать в себя столбец с целевой переменной). Расчет метрик моделей происходит последовательно в одном процессе. Сервер возвращает `CompareModelsResponse`.

11. GET-запрос списка моделей **/models\_list**. Если список обученных моделей пуст, сервер возвращает `RequestError`, иначе – `ModelsListResponse`.

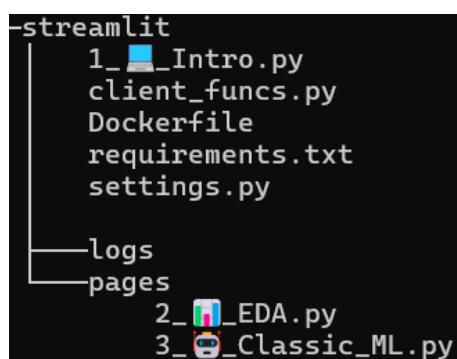
12. DELETE-запрос на удаление модели с заданным ID `/remove_model/{model_id}`. ID модели передается как Path-параметр. Если ID не найден, сервер возвращает RequestError. В противном случае – IdResponse(id=model\_id, status='removed').

13. DELETE-запрос на удаление всех моделей `/remove_all`. Если список моделей изначально был пуст, сервер возвращает RequestError, иначе – список IdResponse(id=model\_id, status='unload').

## 2. Клиентская часть (Streamlit)

### 2.1. Структура программы Streamlit

Структура программы Streamlit представлена на рисунке:



Рассмотрим эту структуру подробнее:

- `1_intro.py` – файл стартовой страницы приложения;
- `pages/2_EDA.py` – файл страницы просмотра разведочного анализа данных;
- `pages/3_Classic_ML.py` – файл страницы обучения и инференса моделей;
- `client_funcs.py` – файл с функциями запросов к серверу, а также с некоторыми функциями, которые не связаны с клиент-серверным взаимодействием, но используются на всех страницах;
- `settings.py` – файл, в котором определена конфигурация программы.

Эмодзи в названии файлов страниц отображаются в приложении в качестве иконок.

### 2.2. Настройки программы

Как в и серверной части, в клиенте используется класс настроек Settings, унаследованный от `pydantic_settings.Settings`. Переменные конфигурации:

- FASTAPI\_URL: str – URL сервера (при запуске локально должен быть равен "http://127.0.0.1:8000/", при запуске в Docker – "http://fastapi:8000/", при запуске на VPS – "https://ai-year-project-service-qeke.onrender.com/");
- ROUTE: str – префикс API-роутера, который используется на сервере;
- GITHUB\_LINK: str – ссылка на репозиторий проекта для подстановки в виджеты.

### **2.3. Функции запросов к серверу (client\_funcs.py)**

Кратко перечислим основные функции запросов к серверу (все эти функции в качестве результата возвращают `requests.Response`):

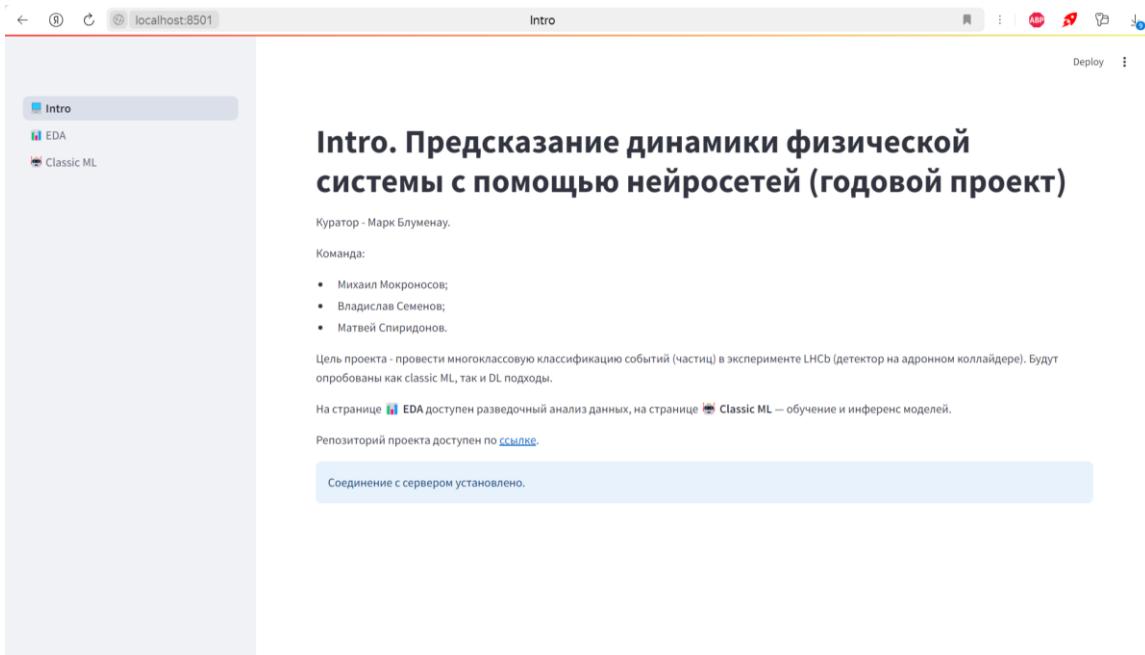
1. **get()** – «корневой» GET-запрос;
2. **get\_pdf()** – запрос файла PDF с выполненным разведочным анализом данных;
3. **get\_columns()** – запрос списка столбцов датасета (используется на тех страницах, где требуется загружать данные, чтобы была возможность сравнить структуру загружаемого датасета с «эталонной» структурой, хранящейся на сервере);
4. **get\_model\_types()** – запрос списка доступных для обучения типов моделей;
5. **train\_models(request\_list: List[Dict], file: UploadedFile)** – запрос на обучение моделей, перечисленных в списке `request_list`, на тренировочном датасете `file`;
6. **get\_current\_model()** – запрос ID текущей модели, которая установлена для инференса;
7. **predict(file: UploadedFile)** – запрос на выполнение предсказаний по файлу с данными `file`.
8. **get\_models\_list()** – запрос списка обученных моделей;
9. **remove\_all()** – запрос на удаление всех обученных моделей;
10. **remove\_model(model\_id: str)** – запрос на удаление модели с заданным ID;
11. **set\_model(model\_id: str)** – запрос на установку модели с заданным ID в качестве текущей модели для инференса;
12. **unset\_model()** – запрос на снятие текущей модели с инференса;
13. **compare\_models(ids: Dict[str, List[str]], file: UploadedFile)** – запрос на сравнение качества моделей, заданных в `ids`, на датасете `file`.

Кроме перечисленных выше функций, `client_funcs.py` содержим функцию `check_dataset(df: pd.DataFrame, cols_data: Dict[str, Any], mode: str = "train") -> bool` для сравнения перечня столбцов датасета `df` и их типов с

«эталонной» информацией о столбцах, полученной с сервера (параметр mode определяет, должен для в датасете быть столбец с целевой переменной). Данная функция не выполняет запросы к серверу.

## 2.4. Интерфейс стартовой страницы приложения

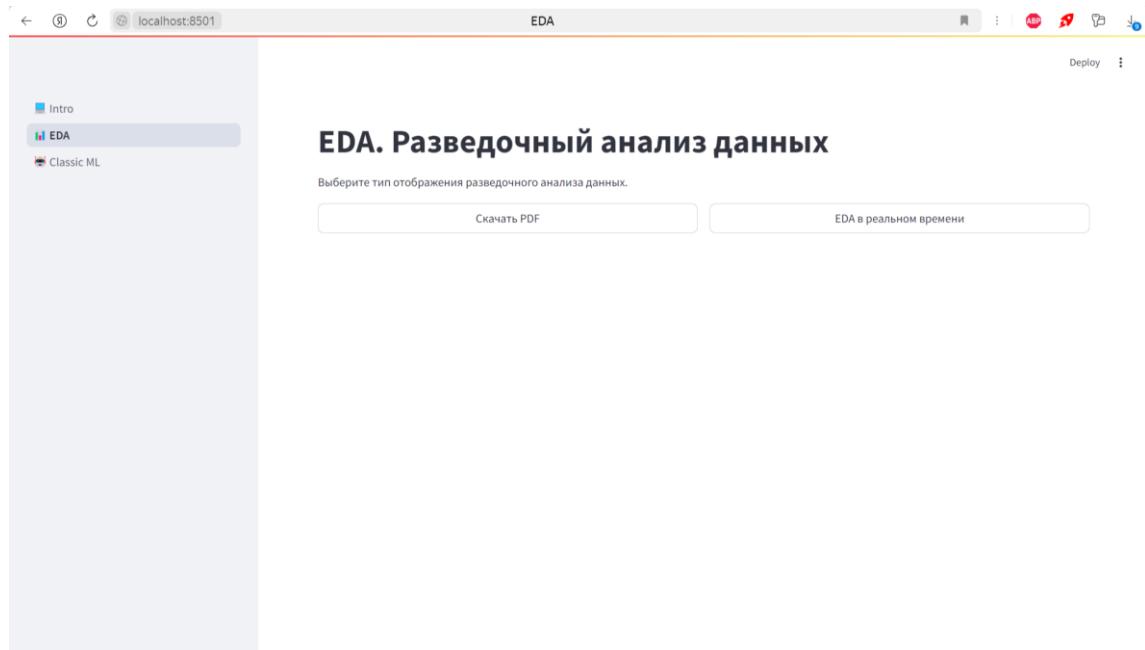
Интерфейс стартовой страницы показан на рисунке.



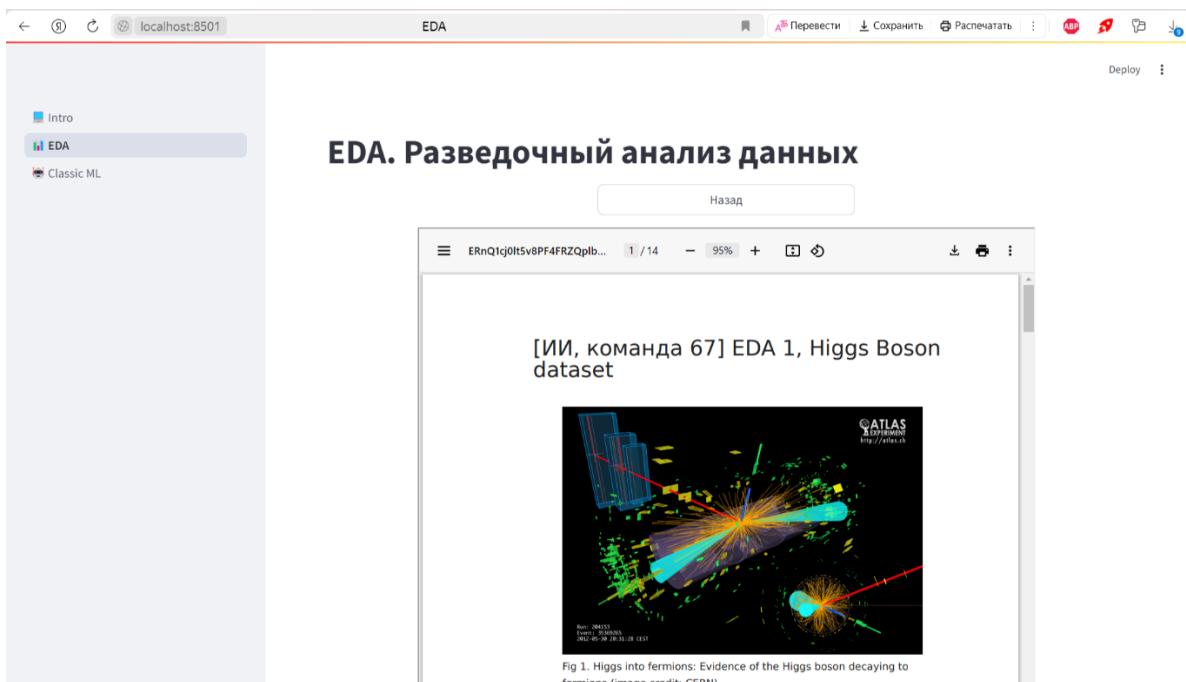
На данной странице выполняется «корневой» GET-запрос для проверки соединения с сервером.

## 2.5. Интерфейс страницы просмотра разведочного анализа данных

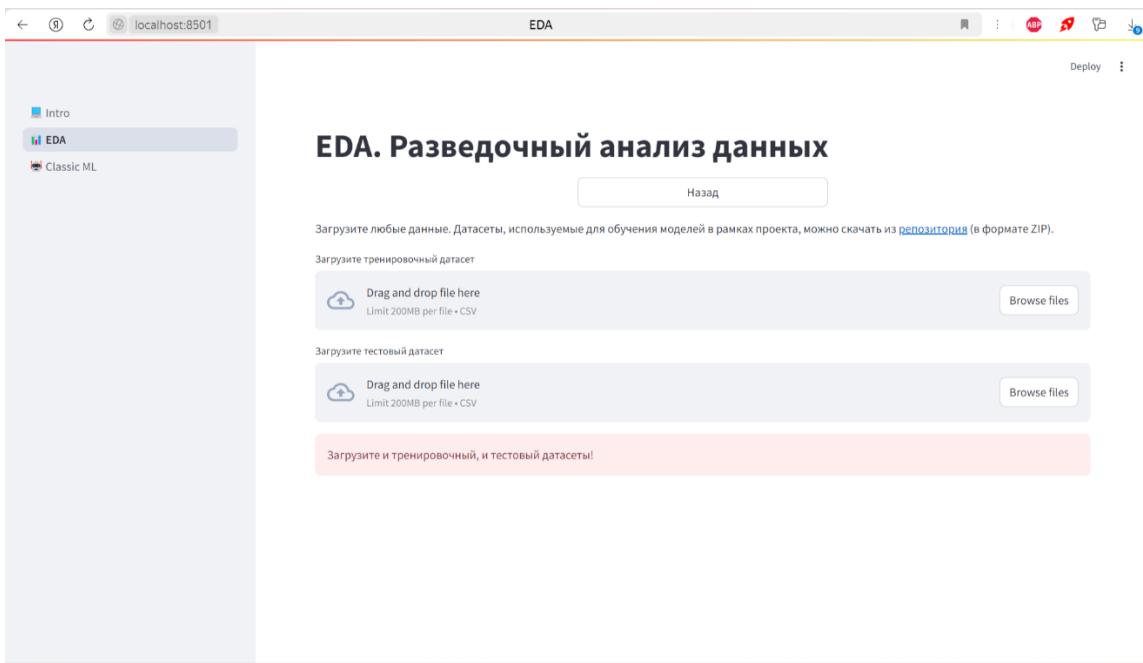
Начальный экран страницы разведочного анализа данных содержит 2 кнопки: для скачивания PDF и отображения анализа данных в реальном времени по загруженным данным:



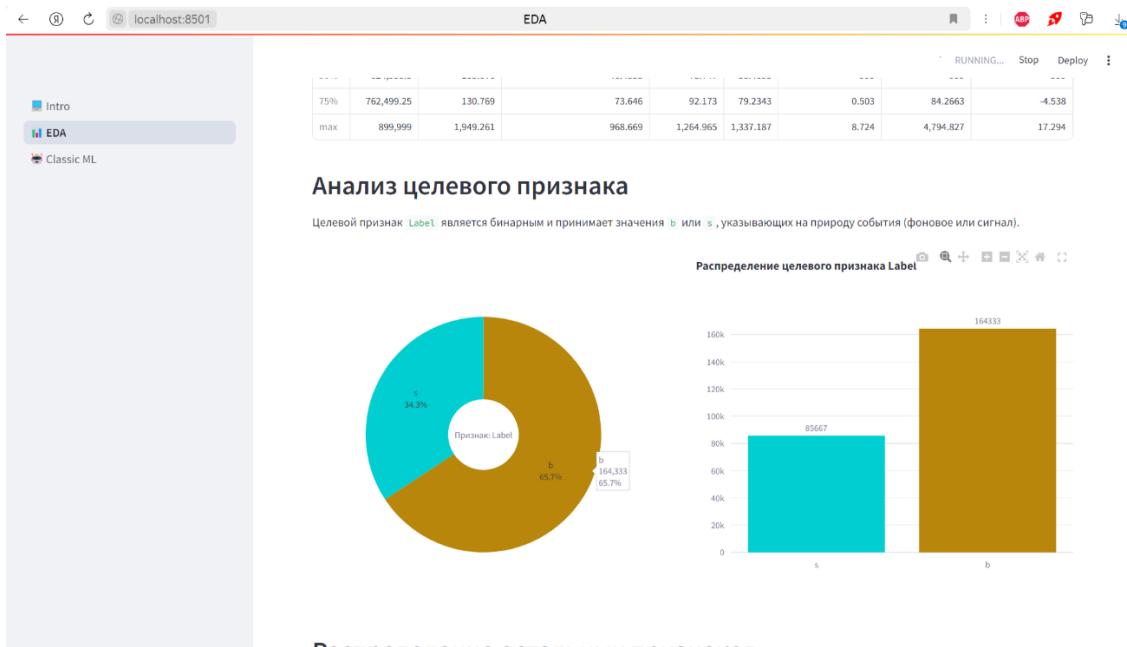
При нажатии кнопки «Скачать PDF» будет открыт виджет с файлом:



В свою очередь, при нажатии кнопки «EDA в реальном времени» откроется интерфейс загрузки тренировочного и тестового датасетов:



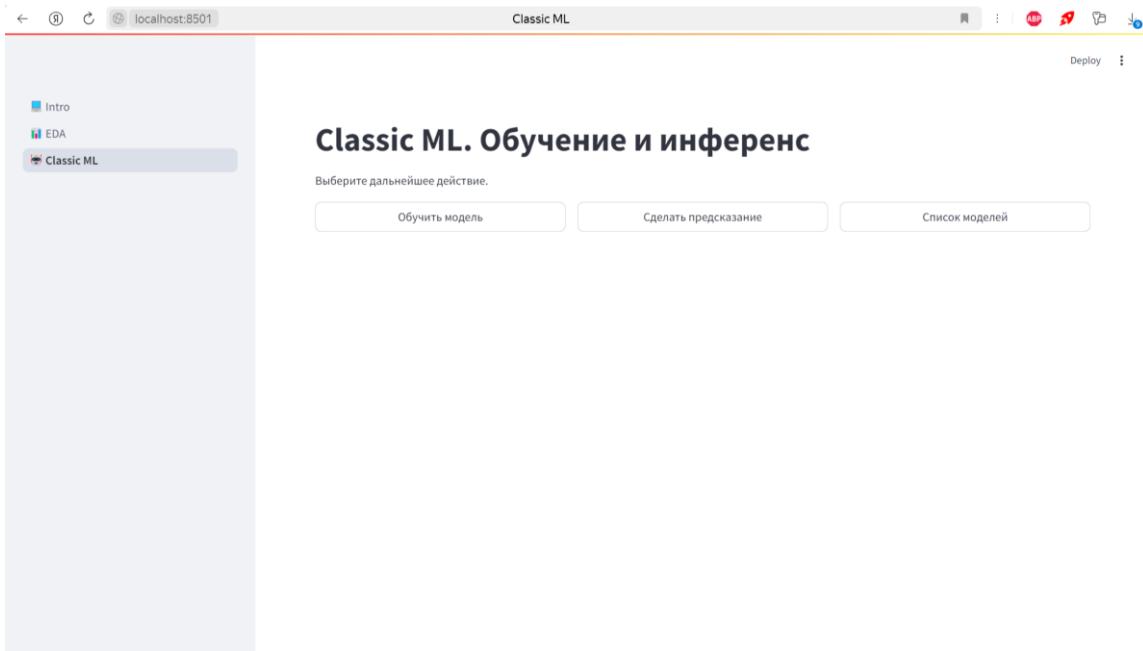
После того, как датасеты будут загружены и проверены на соответствие столбцов и типов, приложение в режиме реального времени отрисует графики (для построения графиков используются библиотеки matplotlib, seaborn и plotly):



Переходы с экрана на экран по кнопкам задаются при помощи переменной streamlit.session\_state.eda\_type.

## 2.6. Интерфейс страницы обучения и инференса моделей

Начальный экран страницы разведочного анализа данных содержит 3 кнопки: для скачивания PDF и отображения анализа данных в реальном времени по загруженным данным:

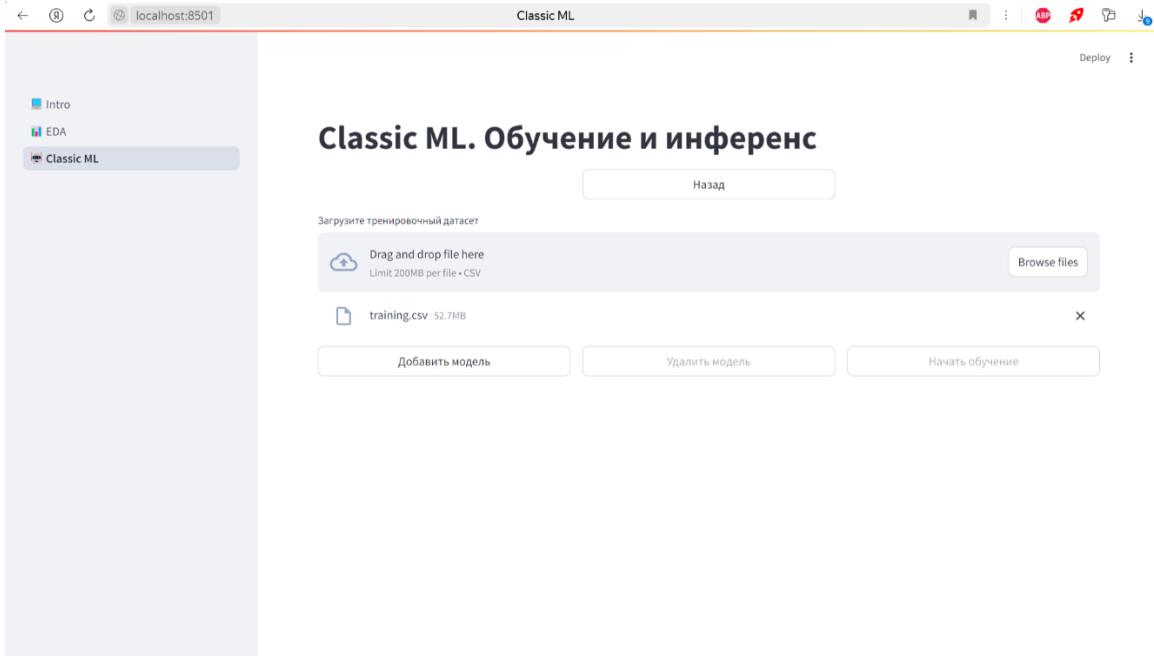


При загрузке начального экрана выполняется запрос `client_funcs.get_columns()` и `client_funcs.get_model_types()`. Для переходов с экрана на экран используются следующие переменные состояния:

- для навигации по кнопкам «Обучить модель», «Сделать предсказание», «Список моделей», «Назад» – переменная `streamlit.session_state.model_task`;
- для навигации в рамках раздела «Обучить модель» – переменная `streamlit.session_state.train_task`;
- для навигации в рамках раздела «Сделать предсказание» – переменная `streamlit.session_state.predict_task`;
- для навигации в рамках раздела «Список моделей» – переменная `streamlit.session_state.list_task`;
- для навигации при сравнении моделей в рамках раздела «Список моделей» – переменная `streamlit.session_state.choose_task`.

### 2.6.1. Кнопка «Обучить модель»

После нажатия кнопки «Обучить модель» будет открыт следующий интерфейс (данные были загружены вручную):



Кнопки «Добавить модель» и «Удалить модель» очевидным образом позволяют нам сформировать список моделей для обучения. Количество обуемых моделей задается переменной `streamlit.session_state.creating_count`. Информация о каждой новой модели сохраняется в следующие переменные:

- `streamlit.session_state[f'mtype_{index}']` – тип модели под номером `index`;
- `streamlit.session_state[f'model_id_{index}']` – ID модели под номером `index`;
- `streamlit.session_state[f'params_{index}']` - перечень гиперпараметров модели под номером `index`;
- `streamlit.session_state[f'ptypes_{index}']` – типы гиперпараметров модели под номером `index`.

ID модели и значение параметров вводится вручную, тип модели и типы гиперпараметров выбираются из предложенных перечней:

The screenshot shows the 'Classic ML' application interface. On the left, there's a sidebar with 'Intro', 'EDA', and 'Classic ML' buttons, where 'Classic ML' is selected. The main area is titled 'Модель 1' (Model 1) and says 'Выберите тип модели:' (Select model type:). A dropdown menu shows 'LogReg'. Below it, a message says 'Введите ID модели' (Enter model ID). A large blue box contains the instruction 'Введите параметры (или оставьте поля пустыми - тогда будут подставлены параметры по умолчанию)' (Enter parameters (or leave fields empty - then default parameters will be used)). There are seven parameter fields with dropdown menus for their types: 'C' (Type: Integer), 'class\_weight' (Type: Integer), 'dual' (Type: Integer), 'fit\_intercept' (Type: Integer), 'intercept\_scaling' (Type: Integer), 'l1\_ratio' (Type: Integer), and 'tol' (Type: Integer).

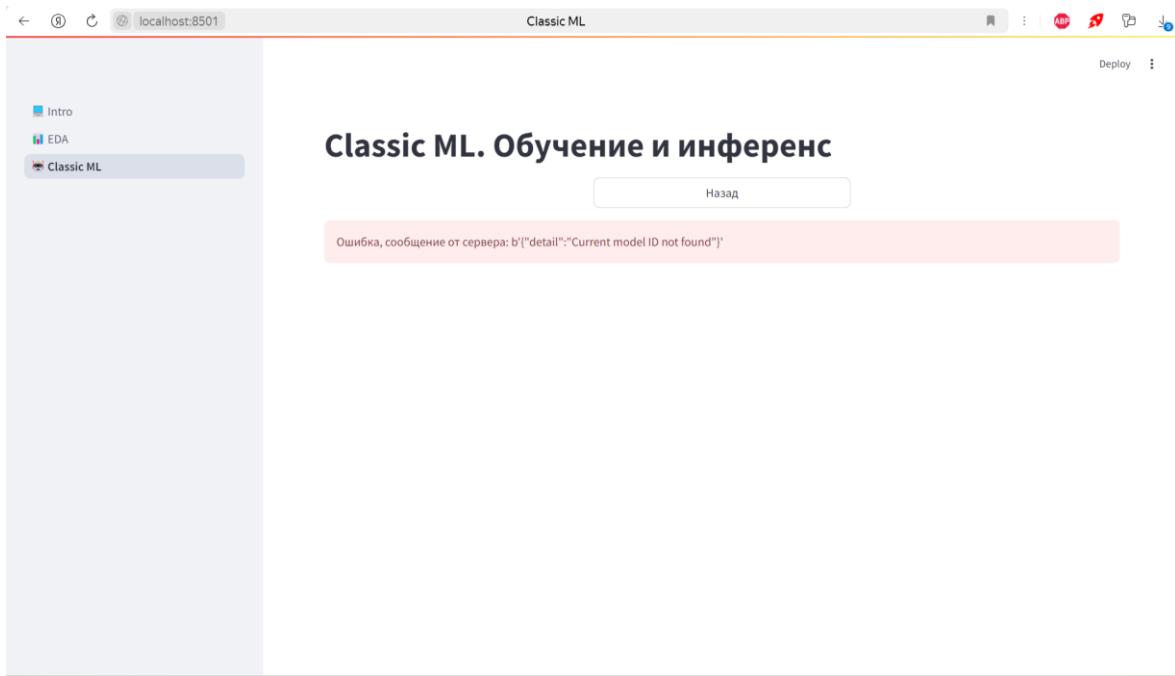
Попробуем обучить модель m1 без ошибок, модель m2, которая обучается слишком долго (SVM), и модель m3 с ошибкой в гиперпараметрах (логистическая регрессия, для которой C = 'zzz'). Результат обучения представлен на рисунке.

The screenshot shows the 'Classic ML. Обучение и инференс' (Classic ML. Training and Inference) page. The sidebar is the same as the previous screenshot. The main area has a 'Назад' (Back) button. It displays three messages in colored boxes: a blue box for 'Модель m1 обучена' (Model m1 trained), a pink box for 'Обучение модели m2 прервано' (Training of model m2 interrupted), and a light red box for 'Указаны неверные параметры для модели m3' (Incorrect parameters specified for model m3).

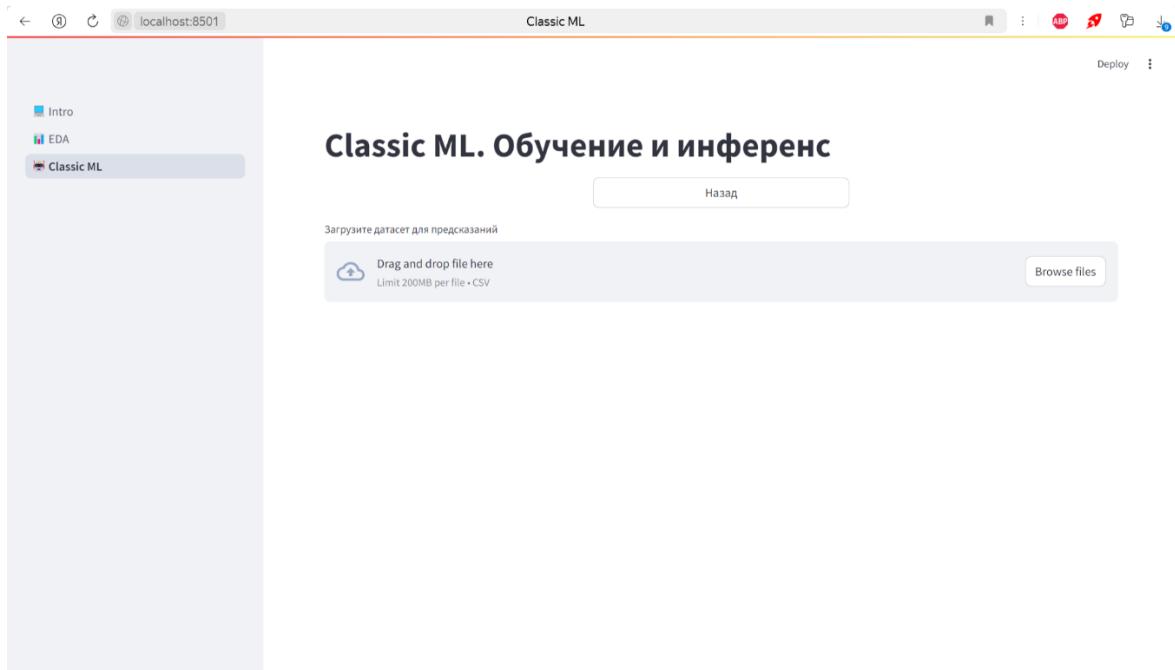
Видно, что для каждого типа ошибок выводится свое сообщение.  
Кнопка «Назад» возвращает пользователя на начальный экран страницы.

## 2.6.2. Кнопка «Сделать предсказание»

Если ни одно модель не установлена для инференса, при нажатии кнопки «Сделать предсказание» приложение отобразит сообщение об ошибке от сервера:



В случае, если модель для инференса установлена, программа предложит загрузить данные:



После загрузки и проверки данных будет отображена таблица с предсказаниями и кнопки скачивания данных в формате CSV:

The screenshot shows a web application interface titled "Classic ML. Обучение и инференс". On the left, there is a sidebar with three items: "Intro", "EDA", and "Classic ML", where "Classic ML" is highlighted with a grey background. The main content area has a header "Classic ML" and a "Deploy" button. Below the header, there is a button "Назад" (Back). A message "Предсказания модели представлены ниже." (Predictions from the model are shown below) is displayed above a table. The table has two columns: "Eventid" and "predictions". The data in the table is as follows:

Eventid	predictions
350,002	0
350,003	0
350,004	0
350,005	0
350,006	0
350,007	0
350,008	0
350,009	1
350,010	0
350,011	0

At the bottom of the table is a "Сохранить" (Save) button.

### 2.6.3. Кнопка «Список моделей»

При нажатии кнопки «Список моделей» первоначально выполняются запросы `client_funcs.get_current_model()` и `client_funcs.get_models_list()`. Экран списка моделей представлен на рисунке:

The screenshot shows the same web application interface as the previous one, but now the main content area displays the title "Список моделей". Below it, there is a section titled "Модель m1" with the subtext "Это текущая модель". This section lists various parameters and their values for the current model:

Тип:	LogReg
C	1.0
class_weight	None
dual	False
fit_intercept	True
intercept_scaling	1
l1_ratio	None
max_iter	100
multi_class	deprecated

Экран списка моделей содержит следующие кнопки:

The screenshot shows a web-based interface titled 'Classic ML'. On the left, there's a sidebar with links: 'Intro', 'EDA', and 'Classic ML' (which is highlighted). The main area displays a table of configuration parameters:

Parameter	Value
l1_ratio	None
max_iter	100
multi_class	deprecated
penalty	l2
random_state	None
solver	lbfgs
tol	0.0001
verbose	0
warm_start	False

Below the table are four buttons in a grid:

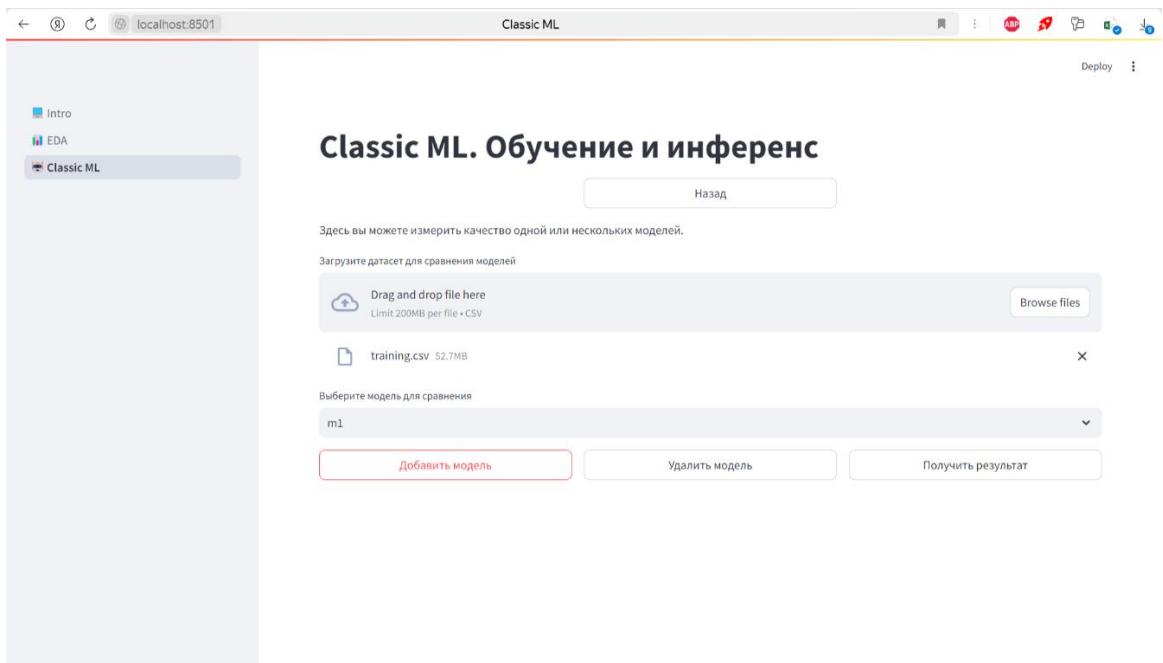
- Установить текущую модель (Set current model)
- Снять текущую модель с инференса (Remove current model from inference)
- Удалить модель (Delete model)
- Удалить все модели (Delete all models)

A final button at the bottom right is 'Сравнить модели' (Compare models).

При нажатии кнопок «Установить текущую модель» и «Удалить модель» открывается идентичный интерфейс, предлагающий выбрать ID модели из списка доступных:

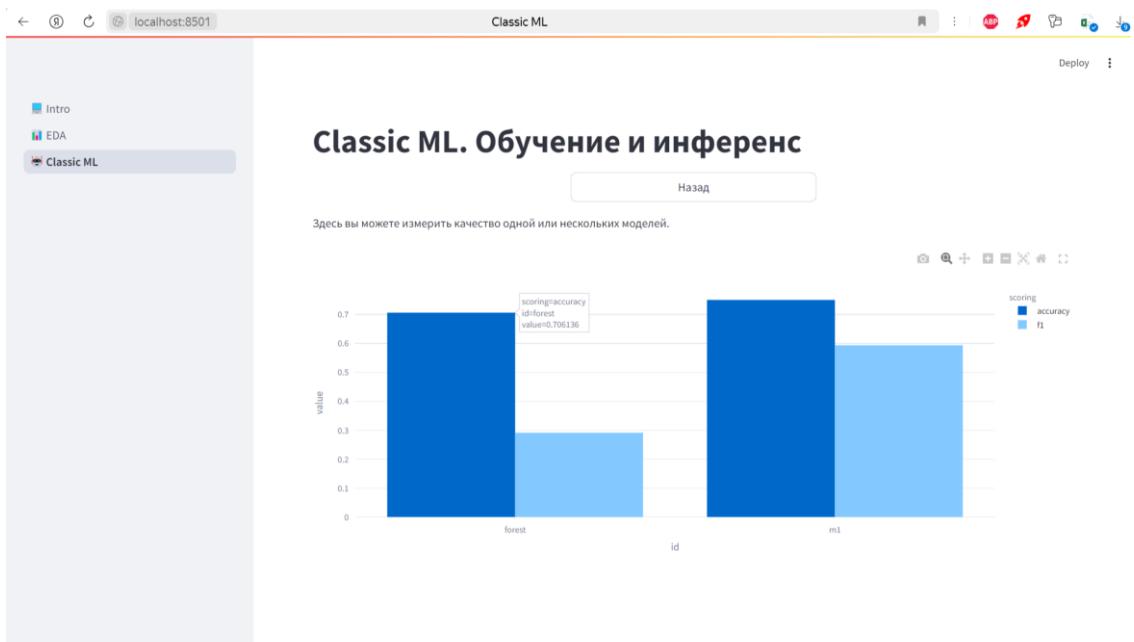
The screenshot shows a sub-interface titled 'Classic ML. Обучение и инференс'. It has a header 'Выберите id модели' (Select model ID) and a dropdown menu containing the value 'm1'. At the bottom are two buttons: 'Принять' (Accept) and 'Назад' (Back).

При нажатии кнопки «Сравнить модели» открывается интерфейс следующего содержания (данные были загружен вручную):



Как и в случае с обучением моделей, по кнопкам «Добавить модель» и «Удалить модель» мы можем сформировать список сравниваемых моделей (в этом случае размер списка определяется переменной `streamlit.session_state.choosing_count`; `id` отобранной для сравнения модели под номером `index` хранится в переменной `streamlit.session_state[f'choosing_id_{index}']`). Для нескольких моделей может быть выбран одинаковый индекс, но график будет построен только один раз.

Пример графика показан на рисунке ниже. Построение графиков выполнялось с помощью `plotly`.



### **3. Развёртывание FastAPI и Streamlit**

#### **3.1. Локальное развертывание**

После скачивания исходников проекта из репозитория и установки зависимостей сервер можно запустить из консоли следующей командой:

```
uvicorn main:app --host 0.0.0.0 --port 8000 --log-config log_config.json --reload
```

В свою очередь, Streamlit-клиент запускается по команде:

```
streamlit run 1_💻_Intro.py
```

**Важно: переменная FASTAPI\_URL в streamlit\settings.py должна быть равна "http://127.0.0.1:8000/".**

#### **3.2. Развёртывание в Docker**

В папке service находится файл docker-compose.yml. Чтобы запустить Docker-контейнеры, необходимо выполнить команду:

```
docker-compose -f docker-compose.yml up
```

**Важно: переменная FASTAPI\_URL в streamlit\settings.py должна быть равна "http://fastapi:8000/".**

#### **3.2. Развёртывание на VPS**

В рамках чекпоинта удалось развернуть сервер и клиент на render.com. Для этого понадобилось по шаблону создать 2 репозитория:

- <https://github.com/Vladislav-IS/ai-year-project-service-fastapi> - сервер;
- <https://github.com/Vladislav-IS/ai-year-project-service-streamlit> - клиент.

Данные репозитории были загружены на сайт render.com и развернуты в качестве веб-сервисов.

The screenshot shows the Render Dashboard interface. On the left, there's a sidebar with 'Projects' selected, showing 'Blueprints' and 'Environment Groups'. Below that is the 'WORKSPACE' section with 'Billing' and 'Settings'. A vertical sidebar on the far left includes links like 'Docs', 'Changelog', 'Community', 'Feedback', 'Invite a friend', 'Compliance and documents', and 'Contact support', along with a 'Render Status' indicator.

The main area is titled 'Overview' under 'Projects'. It features a 'Get organized with Projects' section with a call-to-action button '+ Create your first project' and a 'Learn more' link. To the right is a 'PROJECT Health Checks' panel showing 'All services up' with three tabs: 'PRODUCTION', 'STAGING', and 'DEV'. Below this is a table titled 'Ungrouped Services' with two rows:

	SERVICE NAME	STATUS	RUNTIME	REGION	DEPLOYED	...
<input type="checkbox"/>	ai-year-project-service-fastapi	✓ Deployed	Python 3	Oregon	5h	...
<input type="checkbox"/>	ai-year-project-service-streamlit	✓ Deployed	Python 3	Oregon	5h	...

Из недостатков выбранной платформы VPS следует отметить нестабильность работы развернутых сервисов. Также пришлось избавиться от мультипроцессорности при обучении моделей, поскольку предоставленные мощности не позволили адекватно ее реализовать.

FastAPI-сервер доступен по ссылке: <https://ai-year-project-service-geke.onrender.com>. В свою очередь, Streamlit-клиент доступен по ссылке: <https://ai-year-project-service-streamlit.onrender.com>.