* **Задача 1. Сериализация** — сериализируйте модель. Для этого придётся поправить реализацию пайплайна.
* **Задача 2. API-методы** — реализуйте три API-метода: GET/status, GET/version, POST/prediction.
* **Задача 3 (необязательное задание).Запуск модели по расписанию** — реализуйте запуск модели по расписанию с помощью APScheduler.

Задача 1. Сериализация

Эту модель и её пайплайн обработки данных вы создали, когда выполнили практическую работу для модуля 30 (Deployment).

Ваш пайплайн сложнее того, который мы разбирали в материалах. Вам пришлось писать собственные функции обработки данных и интегрировать их в пайплайн с помощью класса [FunctionTransformer](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.FunctionTransformer.html) из библиотеки sklerarn. В этом заключается основная трудность. К сожалению, пакет joblib не умеет сериализировать обычные функции в pickle-файлы, поэтому просто так сохранить ваш пайплайн не получится. Есть разные способы решить эту проблему. Тут мы предлагаем воспользоваться одним из самых простых — заменить пакет joblib на пакет [dill](https://pypi.org/project/dill/), который умеет сериализовать объекты с очень сложной структурой, включая обычные (свободные) функции.

Интерфейс у dill почти такой же, как у joblib, однако, при сохранении/чтении объектов, в функции dill нужно передавать объекты файлов, а не пути к ним.

Например, сериализовать объект можно так:

with open(file\_name, 'wb') as file:  
   dill.dump(object\_to\_dump, file)

А десериализовать так:

with open(file\_name, 'rb') as file:  
   object\_to\_load = dill.load(file)

Кроме того, каждая свободная функция должна быть автономна — всё, что нужно для исполнения этой функции, должно быть внутри функции. Например, если функция, которая используется в FunctionTransformer (назовем её целевой), вызывает другие свободные функции, то они должны быть определены внутри целевой функции.

Если ваш код удаления выбросов реализован примерно так:

def calculate\_outliers(data):  
    q25 = data.quantile(0.25)  
    q75 = data.quantile(0.75)  
    iqr = q75 - q25  
    boundaries = (q25 - 1.5 \* iqr, q75 + 1.5 \* iqr)  
  
   return boundaries  
  
def remove\_outliers(df):  
    df = df.copy()  
  
    boundaries = calculate\_outliers(df['year'])  
  
    df.loc[df['year'] < boundaries[0], 'year'] = round(boundaries[0])  
    df.loc[df['year'] > boundaries[1], 'year'] = round(boundaries[1])  
  
 return df

то цикл сериализации-десериализации не сработает, потому что свободная функция remove\_outliers, которой вы инициализируете [FunctionTransformer](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.FunctionTransformer.html), вызывает другую свободную функцию calculate\_outliers. Calculate\_outliers не попадёт в итоговый pickle-файл при сериализации такой конструкции и, когда вы будете его использовать, пайплайн не сработает, потому что не сможет найти calculate\_outliers.

Можно исправить положение, изменив код так:

def remove\_outliers(df):  
   def calculate\_outliers(data):  
       q25 = data.quantile(0.25)  
       q75 = data.quantile(0.75)  
       iqr = q75 - q25  
       boundaries = (q25 - 1.5 \* iqr, q75 + 1.5 \* iqr)  
  
       return boundaries  
  
   df = df.copy()  
  
   boundaries = calculate\_outliers(df['year'])  
  
   df.loc[df['year'] < boundaries[0], 'year'] = round(boundaries[0])  
   df.loc[df['year'] > boundaries[1], 'year'] = round(boundaries[1])  
  
  
   return df

Теперь *calculate\_outliers* является частью *remove\_outliers* и будет сериализована вместе с ней.

**Что нужно сделать**

* Проверьте реализацию вашего пайплайна и, если необходимо, поправьте её в соответствии с рекомендациями выше. Для сериализации/десериализации используйте [dill](https://pypi.org/project/dill/).
* Перед тем, как сериализовать итоговый пайплайн, вызовите метод fit на всей обучающей выборке, как мы делали в видео.
* Когда вы переделаете (если необходимо) код вашего пайплайна, запустите его и сохраните пайплайн в файл.

**Советы и рекомендации**

Если совсем запутаетесь, можете подглядывать в [реализацию](https://drive.google.com/file/d/1KpQx9EdD2to-jUffLWhuUS6O46gXkbuA/view?usp=sharing) пайплайна. Однако, настоятельно рекомендуем попытаться разобраться самостоятельно. Инженер — это не просто профессия, это — стиль жизни, основанный на постоянном поиске решения проблем.

Задача 2. API-методы

Теперь приступайте к разработке API с помощью фреймворка [FastAPI.](https://fastapi.tiangolo.com/)

**Что нужно сделать**

Реализуйте следующие методы:

* **GET/status**— возвращает строку I’m OK;
* **GET/version**— возвращает словарь, содержащий информацию о модели (название модели, имя автора, версия, дата создания, тип решающего алгоритма и точность, достигнутая на кросс-валидации), в формате:
* {  
     "name": "Car price prediction model",  
     "author": "Peter Emelianov",  
     "version": 1,  
     "date": "2022-05-31T19:09:57.993322",  
     "type": "SVC",  
     "accuracy": 0.7871  
  }
* **POST/prediction**— принимает JSON-объект, содержащий информацию об автомобиле и возвращающий соответствующее предсказание модели.

Примеры JSON-объектов, с которыми должен работать метод:

* [7316509996.json](https://drive.google.com/file/d/1HaO8bxF4Wbrl5GHlhW2VegWugTpP_FgE/view?usp=sharing);
* [7316152972.json](https://drive.google.com/file/d/1tBNUclPG2bVzH58vl0x1-h0sKUIyYOJ4/view?usp=sharing);
* [7315173150.json](https://drive.google.com/file/d/1_ij3X0ntxIvwyYxLErMnYhFdz1AzGFs5/view?usp=sharing);
* [7313922964.json](https://drive.google.com/file/d/1z5RmMtPFjglGdUau0vHfszIp4cm27aw0/view?usp=sharing);
* [7310993818.json](https://drive.google.com/file/d/1T7G0h14PeMdCvqR2fNeiokNPMlzp_RsK/view?usp=sharing).

Для работы с такими данными создайте pydantic-модель. Не забудьте перечислить все имена атрибутов и их типы данных.

**Советы и рекомендации**

Если не хотите механически перепечатывать вручную, примените творческий подход. Например, код:

df = pd.read\_csv('data/homework.csv').drop('price\_category', axis=1)  
  
  
types = {  
    'int64': 'int',  
    'float64': 'float'  
}  
for k, v in df.dtypes.iteritems():  
    print(f'{k}: {types.get(str(v), "str")}')

 Он сгенерирует правильные имена атрибутов и их типы данных. Останется только скопировать его в класс вашей pydentic-модели в коде. Учитесь решать скучные задачи креативно!

Для ответа, который возвращает обработчик **POST/prediction**, тоже определите модель данных. Ответ должен содержать:

1. Идентификатор (id) автомобиля из входящего JSON.
2. Цену автомобиля (price) из входящего JSON.
3. Предсказанную категорию цены (low, medium, high).

Пример ответа:

{  
   "id": 7313922964,  
   "pred": "high",  
   "price": 65900  
}

Задача 3 (необязательное задание). Запуск модели по расписанию

В качестве задачи «со звёздочкой» предлагаем вам настроить запуск модели по расписанию с помощью модуля APSchedule.

**Что нужно сделать**

Реализуйте скрипт запуска модели по расписанию. Для этого задекорируйте функцию, которая будет считывать пять записей датафрейма и выводить для них предсказания декоратором sched.

Для простоты запуск функции должен производиться каждые пять секунд.

Пример ответа в консоли:

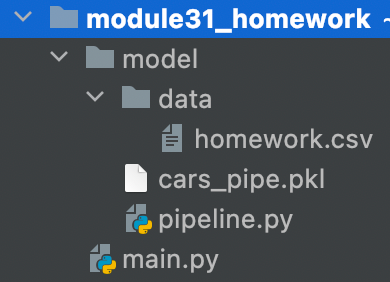
              id  price predicted\_price\_cat  
5673  7312434708   9500                 low  
2372  7314488991  17995              medium  
935   7315635738  11000              medium  
1333  7316673947   6990                 low  
4815  7315316068  16995              medium  
  
  
              id  price predicted\_price\_cat  
2862  7310799134  19637              medium  
3595  7312884179  42998                high  
4003  7309425969  32800                high  
9753  7312967841  14990              medium  
1074  7313952999  19990              medium  
  
  
...

Что оценивается

1. Файловая структура проекта соответствует требуемой.
2. pipeline.py генерирует и сохраняет пайплайн в pickle.
3. main.py загружает модель из pickle и запускает сервис, который отвечает по запросам на пути /status, /version, /predict из задания.
4. Критерий для необязательного задания: schedule.py загружает модель и выдаёт предсказания каждые пять секунд.

Как отправить работу на проверку

Выполните всю работу в формате **одного**проекта Pycharm. Файловая структура проекта такая:



В ней:

* [homework.csv](https://drive.google.com/file/d/14Zzg2DrIfv5DG5UQCM7zVP-SHtz2YxOV/view?usp=sharing) — датасет с данными об автомобилях;
* pipeline.py — файл, который реализует код подготовки пайплайна и обучения модели и сохраняет их в pickle-файл;
* cars\_pipe.pkl — сохранённый (сгенерированный pipeline.py) pickle-файл с пайплайном;
* main.py — реализация API с помощью FastAPI.

Когда закончите, убедитесь, что ваш сервис работает и отвечает на запросы правильно. Для этого используйте программу Postman. Затем целиком заархивируйте папку с проектом и пришлите архив через форму ниже.