

Инференс

[Введение](#)

[Pickle](#)

[Этап 1](#)

[Этап 2](#)

[Этап 3](#)

[Этап 4](#)

[Этап 5](#)

[Этап 6](#)

[Ограничения](#)

[streamlit](#)

[Что такое `streamlit`?](#)

[Демо](#)

[Места посадки Uber в Нью-Йорке](#)

[Создание приложения](#)

[Виджеты выбора](#)

[Кэширование](#)

[Визуализация](#)

[Write](#)

[Приложение](#)

[Пример `streamlit` приложения](#)

Введение

Намного дешевле хранить данные в виде готовой модели, чем каждый раз заново обучать модель на сервере.

Именно поэтому код, который был написан в процессе обучения, крайне редко используется для его инференса (от англ inference — вывод). Так называется непрерывная работа алгоритма машинного обучения в конечном приложении. Именно поэтому при внедрении моделей в продакшн их принято сохранять в готовом виде.



Что из себя представляет обученная модель?

По своей сути, как и всё (ну, почти всё) в языке программирования *Python*, она является **объектом**. Этот объект не является простым, ведь ваша модель содержит сложную иерархию классов, в каждом классе есть набор полей, ссылающихся на объекты других классов и так далее.

Для того, чтобы гарантировать сохранение всей структуры данных и получить её при загрузке обратно, используется сериализация.

Сериализация — процесс трансформации любой структуры данных, поддерживаемой в языке, в последовательность битов. Обратной к операции сериализации является операция десериализации.

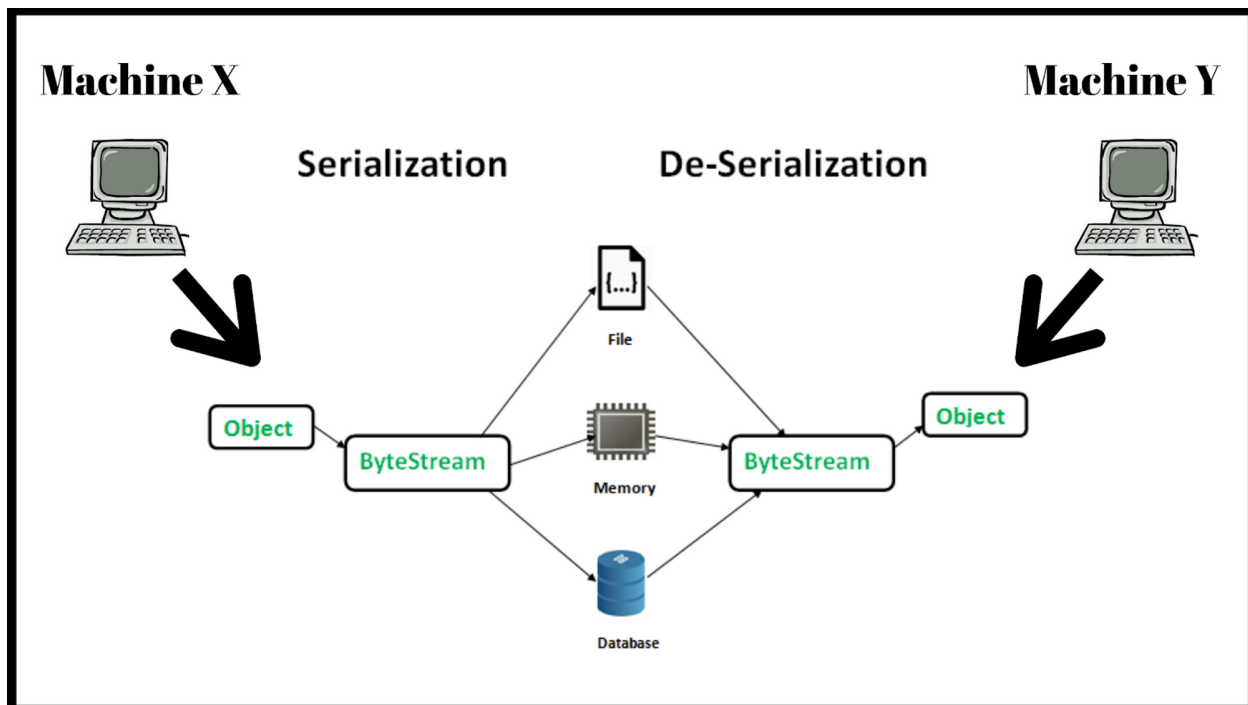
Pickle

Посмотрим, с помощью каких средств происходит сериализация объектов в *Python*.

В стандартную библиотеку *Python* входит модуль Pickle, который служит для сериализации почти всех объектов произвольного типа.

Мы помним, что объекты находятся в оперативной памяти и направляются в байтовые потоки ввода-вывода. В байтовые потоки может быть направлен любой файлоподобный объект.

В ходе десериализации исходный объект воссоздается в оперативной памяти с теми же самыми значениями, но с новой идентичностью — новым адресом в памяти.



Обратите внимание на предупреждение в официальной документации:

Warning. The pickle module is not secure. Only unpickle data you trust.

Так как законсервирован может быть абсолютно любой объект, то в нём могут быть «спрятаны» различные вредоносные программы или данные. Поэтому будьте внимательны и не проводите десериализацию бинарных файлов, в происхождении которых вы не уверены.

Для иллюстрации работы модуля *Pickle* последовательно пройдемся по всем этапам.

Этап 1

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_diabetes

X, y = load_diabetes(return_X_y=True)
regressor = LinearRegression()
regressor.fit(X, y)
```

Объект `regressor` теперь является обученной моделью.

Этап 2

Импортируем модуль и воспользуемся методом `dumps`:

```
import pickle

model = pickle.dumps(regressor)
print(type(model), type(regressor))
```

Этап 3

```
regressor_from_bytes = pickle.loads(model)
```

Этап 4

Сохраним объект прямо в файл:

```
with open('myfile.pkl', 'wb') as output:
    pickle.dump(regressor, output)
```

Теперь у нас есть бинарный файл с готовой моделью.

Этап 5

Файл `myfile.pkl` так же легко десериализовать:

```
with open('myfile.pkl', 'rb') as pkl_file:
    regressor_from_file = pickle.load(pkl_file)

regressor_from_file
```

Этап 6

Убедимся, что методы и результаты предсказаний обученной модели и загруженной из файла совпадают:

```
all(regressor.predict(X) == regressor_from_bytes.predict(X))
all(regressor.predict(X) == regressor_from_file.predict(X))
```

Ограничения

Как мы упоминали, у Pickle есть ограничения. Например, мы не сможем сериализовать лямбда-функции. Давайте посмотрим, что нам вернёт следующий код:

```
import pickle
my_lambda = lambda x: x*2
with open('my_lambda.pkl', 'wb') as output:
    pickle.dump(my_lambda, output)
```

Ошибка: "`PicklingError: Can't pickle <function <lambda>`"

В таких случаях лучше пользоваться пакетом `dill`.

`streamlit`

Что такое `streamlit` ?

Если очень кратко — это библиотека, которая позволяет быстро строить интерактивный интерфейс, пользуясь готовыми блоками (почти как *Tilda* для разработки сайтов). Большой плюс этого интерфейса в том, что впоследствии его можно загрузить на удаленный сервер. Он позволяет создавать стильные приложения благодаря буквально нескольким строкам кода.

Несколько причин использовать инструменты `streamlit`:

- Поддерживают разработку на Python — не нужно знать HTML!
- Для создания прекрасных приложений не нужно много кода.
- Нет нужды в обратных вызовах, так как виджеты обрабатываются как переменные.
- Кэширование данных упрощает и ускоряет вычислительные конвейеры.

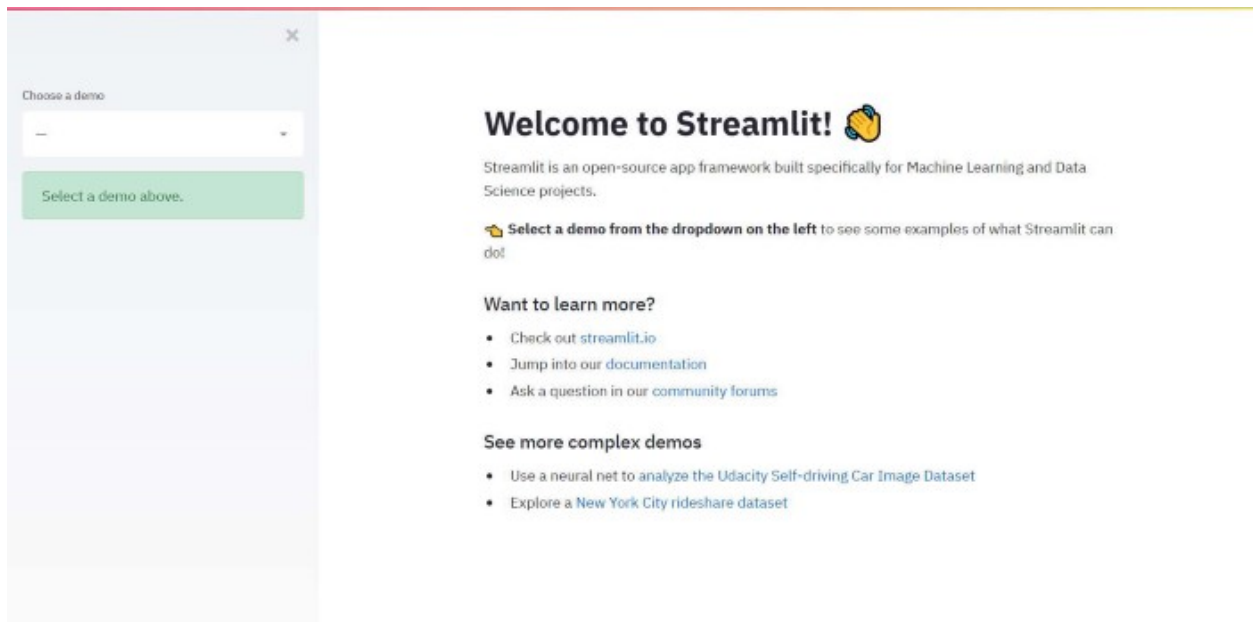
`streamlit` легко установить при помощи следующей команды:

```
pip install streamlit
```

Демонстрацию приложения с примером кода можно посмотреть, используя эту команду:

```
streamlit hello
```

Откроется страница приветствия:



Демо

Над демо `streamlit hello` есть несколько более сложных демо, доступных онлайн. Я перечислю несколько, включая мое, чтобы дать вам представление о возможностях `streamlit`.

Места посадки Uber в Нью-Йорке

Это демо Streamlit показывает, как можно интерактивно визуализировать места посадки Uber в Нью-Йорке.

Uber Pickups in New York City

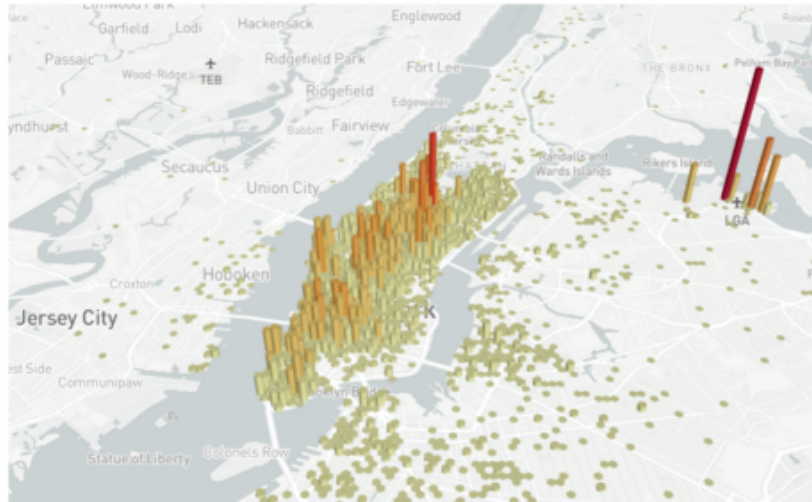
This is a demo of a Streamlit app that shows the Uber pickups geographical distribution in New York City. Use the slider to pick a specific hour and look at how the charts change.

[See source code](#)

Hour to look at



Geo data between 17:00 and 18:00



Просто запускаем код ниже после установки `streamlit`:

```
streamlit run https://raw.githubusercontent.com/streamlit/demo-uber-nyc-pickups/master/streamlit_app.py
```

Создание приложения

Давайте рассмотрим несколько базовых функций, которые вы наверняка будете использовать в своих приложениях.

Виджеты выбора

Одна из основных фиш `streamlit` — использование виджетов. Вот некоторые из доступных:

- **SelectBox**

```
age = streamlit.selectbox("Choose your age: ", np.arange(18, 66, 1))
```

Choose your age:

18

- **Slider**

```
age = streamlit.slider("Choose your age: ", min_value=16,
                        max_value=66, value=35, step=1)
```

Choose your age:



- **MultiSelect**

```
artists = st.multiselect("Who are your favorite artists?",
                          ["Michael Jackson", "Elvis Presley",
                           "Eminem", "Billy Joel", "Madonna"])
```

Who are your favorite artists?

Кэширование

У многих инструментов дашбордов есть проблема — данные загружаются каждый раз при выборе опции или при переключении между страницами. К счастью, в `streamlit` есть удивительная опция, позволяющая кэшировать данные и запускать их, только если они не были запущены ранее.


```
import pandas as pd
import streamlit as st

@st.cache_data
def load_data():
    df = pd.read_csv("your_data.csv")
    return df

# Выполняется, если кэшировано.
df = load_data()
```

В коде выше видно, как можно кэшировать любую созданную функцию, включая загрузку данных, предварительную обработку данных и подготовку сложной модели.

Документацию можно найти [здесь](#).

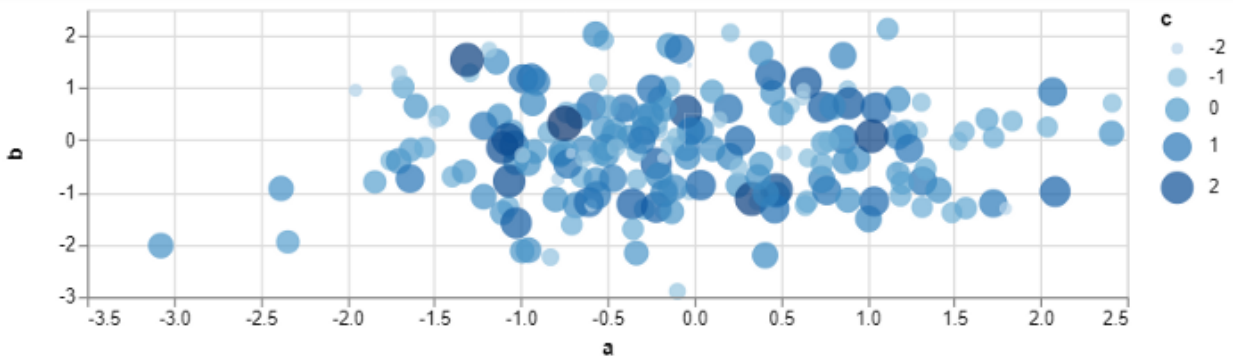
Визуализация

Streamlit поддерживает множество библиотек визуализации, включая Matplotlib, Altair, Vega-Lite, Plotly, Bokeh, Deck.GL и Graphviz. Он может загружать даже **аудио** и **видео**!

Ниже короткий пример отображения графика в Altair:

```
import pandas as pd
import numpy as np
import altair as alt
import streamlit as st

df = pd.DataFrame(np.random.randn(200, 3), columns=['a', 'b', 'c'])
c = alt.Chart(df).mark_circle().encode(x='a', y='b', size='c',
                                       color='c')
st.altair_chart(c, width=-1)
```



Write

Функция `write` — это швейцарский армейский нож команд `streamlit`. Она ведет себя по-разному в зависимости от добавленного аргумента. Например, если вы добавите фигуру `Matplotlib`, она автоматически покажет вам эту визуализацию.

Несколько примеров:

- `write(string)` : выводит отформатированную Markdown строку.
- `write(data_frame)` : отображает DataFrame как таблицу.
- `write(dict)` : отображает словарь в интерактивном виджете.
- `write(keras)` : отображает модель Keras.
- `write(plotly_fig)` : отображает фигуру Plotly.

```
import streamlit as st
import pandas as pd

df = pd.read_csv("https://github.com/MaartenGr/boardgame/raw/master/files/boardgame_new.csv").head()
st.write(df)
```

Приложение

Чтобы показать, как развернуть ваше приложение, для примера создадим простое демо с двумя страницами. На домашней странице будут отображаться выбранные нами данные, а на странице **Exploration** можно визуализировать переменные в графике Altair.

Код ниже выдает `Selectbox` на боковой панели, в котором можно выбрать страницу. Данные закешированы, поэтому нет необходимости постоянно их перезагружать.

```
from vega_datasets import data
import streamlit as st
import altair as alt

def main():
    df = load_data()
    page = st.sidebar.selectbox("Choose a page", ["Homepage", "Exploration"])

    if page == "Homepage":
        st.header("This is your data explorer.")
        st.write("Please select a page on the left.")
        st.write(df)
    elif page == "Exploration":
        st.title("Data Exploration")
        x_axis = st.selectbox("Choose a variable for the x-axis", df.columns, index=3)
        y_axis = st.selectbox("Choose a variable for the y-axis", df.columns, index=4)
        visualize_data(df, x_axis, y_axis)

    @st.cache_data
```

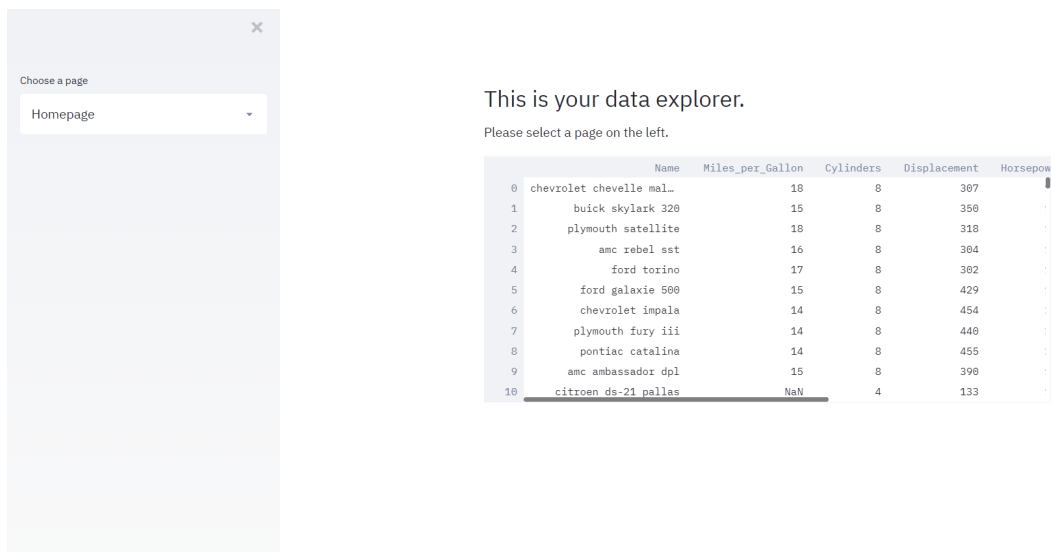
```
def load_data():
    df = data.cars()
    return df

def visualize_data(df, x_axis, y_axis):
    graph = alt.Chart(df).mark_circle(size=60).encode(
        x=x_axis,
        y=y_axis,
        color='Origin',
        tooltip=['Name', 'Origin', 'Horsepower', 'Miles_per_Gallon']
    ).interactive()

    st.write(graph)

if __name__ == "__main__":
    main()
```

Запуск кода `streamlit run app.py` покажет следующую страницу:



Choose a page

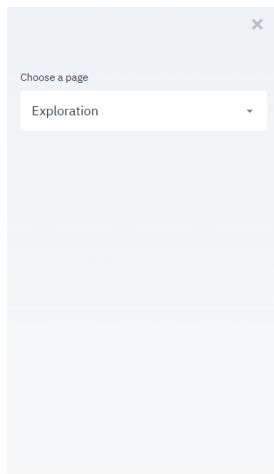
Homepage

This is your data explorer.

Please select a page on the left.

	Name	Miles_per_Gallon	Cylinders	Displacement	Horsepower
0	chevrolet chevelle malibu	18	8	307	
1	buick skylark 320	15	8	350	
2	plymouth satellite	18	8	318	
3	amc rebel sst	16	8	304	
4	ford torino	17	8	302	
5	ford galaxie 500	15	8	429	
6	chevrolet impala	14	8	454	
7	plymouth fury iii	14	8	440	
8	pontiac catalina	14	8	455	
9	amc ambassador dpl	15	8	390	
10	citroen ds-21 pallas	NaN	4	133	

Выбор страницы Exploration покажет следующую визуализацию:



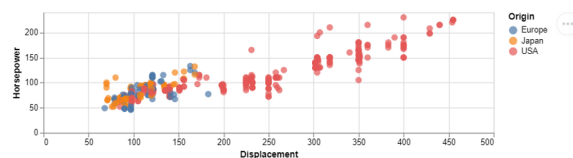
Data Exploration

Choose a variable for the x-axis

Displacement

Choose a variable for the y-axis

Horsepower



Пример `streamlit` приложения

Здесь вы можете найти репозиторий с полным кодом приложения на `streamlit`.