Оглавление

Часть I.
Создание простых приложений Streamlit 9

Глава 1. Введение в Streamlit 10

Почему именно Streamlit? 10
Установка Streamlit 12
Организация приложений Streamlit 12
Демонстрация построения графиков с помощью Streamlit 13
Создание приложения с нуля 15
Использование пользовательского ввода в приложениях Streamlit 21
Последние штрихи: добавляем текст в Streamlit 25
Выводы 28

Глава 2. Загрузка, скачивание и работа с данными 29

Исследуем набор Пингвины Палмера 30
Управление потоком в Streamlit 40
Отладка приложений Streamlit 43
Разработка в Streamlit 44
Исследуем в Jupyter, а затем копируем в Streamlit 44
Операции с данными в Streamlit 45
Пробуем Altair 47
Введение в кеширование 48
Сохранение состояния сеанса (session state) 55
Выводы 59

Глава 3. Визуализация данных 60

Новый набор данных Деревья Сан-Франциско 61
Варианты использования визуализации Streamlit 62
Встроенные графические функции Streamlit 63
Встроенные возможности визуализации в Streamlit 68
Plotly 68
Matplotlib и Seaborn 70
Bokeh 72
Altair 74
PyDeck 76
Выводы 83

Глава 4. Машинное обучение с помощью Streamlit 84

Стандартный процесс создания модели машинного обучения 85
Прогнозирование видов пингвинов 85
Использование предварительно обученной модели машинного обучения в Streamlit 89

Обучение моделей внутри приложений 94
Понимание результатов машинного обучения 98
Интеграция внешней библиотеки машинного обучения в Streamlit на примере библиотеки Transformers от Hugging Face 110
Интеграция внешней библиотеки машинного обучения в Streamlit на примере OpenAI 112
Аутентификация с помощью OpenAI 112
Стоимость API OpenAI 112
Streamlit и OpenAI 113
Выводы 119

Глава 5. Развертывание приложений с помощью Streamlit Community Cloud 120

Начало работы с Streamlit Community Cloud 121
Краткое руководство по GitHub 121
Развертывание с помощью Streamlit Community Cloud 126
Отладка Streamlit Community Cloud 130
Секреты Streamlit 130
Выводы 134

Часть II. Создание сложных приложений Streamlit 135

Глава 6. Улучшение интерфейса приложений Streamlit 136

Начало работы с набором данных Деревья Сан-Франциско 136
Работа с колонками в Streamlit 137
Настройка конфигурации страницы 143
Использование боковой панели Streamlit 144
Выбор цвета с помощью палитры цветов 149
Использование тем Streamlit 150
Многостраничные приложения 153
Редактируемые датафреймы 156
Выводы 161

Глава 7. Знакомство с компонентами Streamlit 162

Обмен программным кодом с помощью streamlit-embedcode 164
Редактирование датафреймов с помощью streamlit-aggrid 167
Детализация графиков Plotly с помощью streamlit-plotly-events 169
Размещение анимаций в приложениях с помощью streamlit-lottie 171
Разведочный анализ с помощью streamlit-pandas-profiling 176
Создание интерактивных карт с помощью streamlit-folium 181
Использование вспомогательных мини-функций с помощью streamlit-extras 184
Поиск дополнительных компонентов 186
Выводы 187

Глава 8. Развертывание Streamlit-приложений с помощью Hugging Face 188

Выводы 197

Глава 9. Подключение к базе данных Snowflake 198

Подключение к Snowflake с помощью Streamlit 199
Улучшение организации подключений 201
Улучшение организации запросов и интерфейса приложения 203
Выводы 204

Часть III. Примеры использования Streamlit 205

Глава 10. Использование Streamlit в пет-проектах и тестовых заданиях для соискателей 206

Использование Streamlit в качестве доказательства, подтверждающего наличие навыков работы с данными 206
Машинное обучение – приложение Пингвины Палмера 207
Визуализация – приложение Прекрасные деревья 209
Использование Streamlit в тестовых заданиях при приеме на работу 210
Задания 211
Выполнение задания 1 212
Выполнение задания 2 220
Выводы 223

Глава 11. Прототипирование проектов в Streamlite 224

Идеи для проектов data science 225
Сбор и чистка данных 227
Создание MVP 229
Сколько книг я читаю каждый год? 229
Сколько времени мне потребуется, чтобы закончить читать начатую книгу? 231
Каков объем книг, которые я прочитал? 235
Каков «возраст» книг, которые я прочитал? 237
Какой рейтинг я ставлю книгам в сравнении с другими пользователями Goodreads? 241
Итеративное улучшение 246
Улучшение внешнего вида с помощью анимации 247
Организация приложения с использованием широкого формата, дополнительных статистик, колонок и сопроводительного текста 250
Хостинг и продвижение 254
Выводы 255

Глава 12. Использование библиотеки прогнозирования временных рядов ETNA в Streamlit 256

Пишем программный код приложения 256

Работа с приложением 279
Загрузка и визуализация данных 279
Определение горизонта прогнозирования 282
Преобразования зависимой переменной 283
Конструирование признаков 284
Список экземпляров классов, выполняющих преобразования зависимой переменной и создающих признаки 285
Итоговый набор 285
Обучение базовой модели 286
Оценка качества и визуализация прогнозов базовой модели 286
Перекрестная проверка 287
Оценка качества и визуализация прогнозов по итогам перекрестной проверки 289
Оптимизация гиперпараметров 290
Получение прогнозов для новых данных 294
Запись прогнозов для новых данных в CSV-файл 295
Работа с экзогенными переменными 295
Подготавливаем наборы с экзогенными переменными 295
Прогнозируем, используя подготовленные наборы экзогенных переменных 300
Развертывание приложения на платформе Streamlit Community Cloud 302

Глава 13. Интервью с опытными пользователями Streamlit 305

Интервью #1 – Фанило Андрианасоло (интервью 2021 года, для первого издания) 306
Интервью #2 – Фанило Андрианасоло (интервью 2023 года) 312
Интервью #3 – Йоханнес Рике 317
Интервью #4 – Эдриен Трейль (интервью 2021 года, для первого издания) 324
Интервью #5 – Эдриен Трейль (интервью 2023 года) 328
Интервью #6 – Чарли Варнье 332
Интервью #7 – Джерард Бентли 337
Интервью #8 – Арно Мирибель и Закари Блэквуд 341
Интервью #9 – Юитиро Татибана 348
Выводы 353

Часть I

Создание простых приложений Streamlit

Глава 1

Введение в Streamlit

Streamlit – это платформа, которая помогает создавать и разрабатывать веб-приложения на основе Python. Веб-приложения можно использовать для обмена результатами аналитики, создания сложных интерактивных приложений и демонстрации новых моделей машинного обучения. Кроме того, разработка и развертывание приложений Streamlit происходят невероятно быстро и гибко, часто уменьшая время разработки приложений с нескольких дней до пары часов.

В этой главе мы начнем с основ Streamlit. Мы научимся скачивать и запускать демонстрационные приложения Streamlit, редактировать демонстрационные приложения с помощью нашего собственного текстового редактора, организовывать наши приложения Streamlit и, наконец, создавать свои собственные. Затем изучим основы визуализации данных в Streamlit. Мы узнаем, как принять исходные пользовательские данные, а затем добавить последние штрихи к нашим собственным приложениям с помощью текста. К концу этой главы вы сможете начать создавать свои собственные приложения Streamlit!

В частности, мы затронем следующие темы:

- почему именно Streamlit?
- установка Streamlit;
- организация приложений Streamlit;
- демонстрация построения графиков с помощью Streamlit;
- создание приложения с нуля.

Почему именно Streamlit?

В течение последнего десятилетия специалисты по data science становятся все более ценным ресурсом для компаний и некоммерческих организаций. Они помогают принимать решения на основе данных, повышать эффективность процессов и внедрять модели машинного обучения для улучше-

ния этих решений в воспроизводимом масштабе. Одна из проблемных точек для специалистов по обработке данных возникает в процессе работы сразу после того, как они нашли новый инсайт или создали новую модель. Как специалистам по данным лучше всего показать динамический результат, новую модель или сложную аналитику? Они могут отправить статическую визуализацию, которая работает в некоторых случаях, но не работает в случае сложных анализов, построенных друг на друге или на чем-то, что требует ввода данных пользователем. Они могут создать документ Word (или экспортировать свою тетрадку Jupyter в виде документа), который сочетает в себе текст и визуализацию, что также не работает для пользовательского ввода и его сложнее воспроизвести. Другой вариант – создать целое веб-приложение с нуля, используя фреймворк, такой как Flask или Django, а затем выяснять, как развернуть все приложение в AWS или другом облачном провайдере. Ни один из этих вариантов не является удобным. Многие из них медленные, не принимают пользовательский ввод или не оптимальны для информирования о процессе принятия решений, столь важного для *data science*.

Познакомьтесь со Streamlit. Streamlit – это все, что связано со скоростью и взаимодействием. Это фреймворк веб-приложений, который помогает вам создавать веб-приложения на Python. Он имеет встроенные и удобные методы для обработки пользовательского ввода, построения графиков с использованием самых популярных и мощных библиотек построения графиков на Python и быстрого развертывания графиков в веб-приложении.

Последний год я потратил на создание самых разных приложений Streamlit, начиная с проектов для моего личного портфолио и заканчивая созданием быстрых приложений для решения домашних задач в области data science и даже мини-приложений для воспроизводимого анализа на работе. Я искренне верю, что Streamlit может быть таким же ценным для вас и вашей работы, как и для моей. Я написал эту книгу, чтобы быстро ввести вас в курс дела, чтобы вы могли ускорить процесс обучения и научились создавать веб-приложения за считанные минуты и часы, а не в течение нескольких дней. Если вам такой путь подходит, читайте дальше. Книга состоит из трех частей. Первая часть начинается с введения в Streamlit и заканчивается созданием ваших собственных базовых приложений Streamlit. Во второй части мы распространим эти знания на более продвинутые темы, такие как методы производственного развертывания и использование компонентов, созданных сообществом Streamlit, чтобы создать еще более красивые и удобные приложения Streamlit. В последней части мы сосредоточимся в основном на практиках опытных пользователей, которые используют Streamlit на работе, в академических кругах и для изучения методов data science. Прежде чем мы начнем, нужно настроить Streamlit и обсудить, как будут организованы примеры, приведенные в этой книге.

Установка Streamlit

Чтобы запускать приложения Streamlit, нужно сначала установить Streamlit. Для этого я использовал менеджер пакетов под названием pip, но вы можете установить его с помощью любого менеджера пакетов по вашему выбору (например, brew). В этой книге используется Streamlit версии 1.24 и Python 3.10.

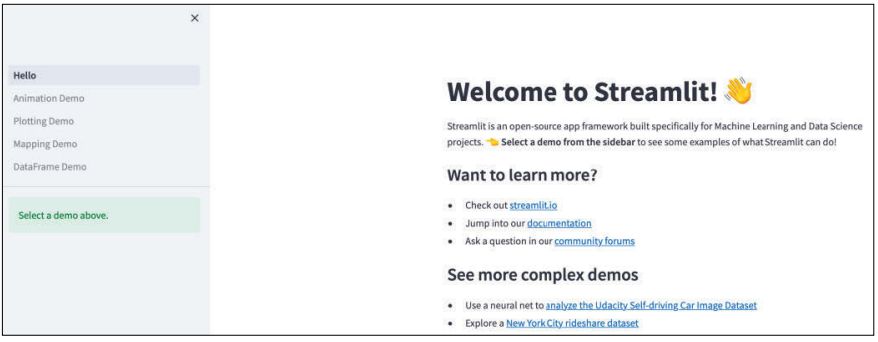
На протяжении всей книги мы будем использовать как команды терминала, так и код, написанный в скриптах Python. В каждом случае мы будем указывать, откуда запускать код. Чтобы установить Streamlit, запустите в терминале код:

```
pip install streamlit
```

Теперь запустите следующий код:

```
streamlit hello
```

И в вашем браузере откроется страница-приглашение Streamlit.



Организация приложений Streamlit

Каждое приложение Streamlit, которое мы создаем в этой книге, должно находиться в отдельной папке. Заманчиво создавать новые файлы для каждого приложения Streamlit, но это способствует развитию вредной привычки, негативные последствия которой скажутся позже, когда мы будем говорить о развертывании приложений Streamlit.

Для этой книги я бы порекомендовал вам иметь специальную отдельную папку, где будут храниться все приложения, которые вы создадите в этой книге. Я назвал свою папку `streamlit_apps`. Следующая команда создаст новую папку с именем `streamlit_apps` и сделает ее нашим текущим рабочим каталогом:

```
mkdir streamlit_apps
cd streamlit_apps
```

Демонстрация построения графиков с помощью Streamlit

Давайте научимся создавать приложения Streamlit. Для этого мы выполняем следующие операции.

1. Создаем Python'овский файл, в котором разместим весь наш код для приложения Streamlit.
2. Вносим небольшие правки для практики.
3. Запускаем наш файл локально.

Первый шаг – создать папку с именем `plotting_app`, в которой будет размещен наш первый пример. Следующий код создает эту папку, изменяет наш рабочий каталог на `plotting_app` и создает пустой файл Python под названием `plot_demo.py`:

```
mkdir plotting_app
cd plotting_app
touch plot_demo.py
```

Теперь, когда мы создали файл `plot_demo.py`, откройте его в любом текстовом редакторе (если у вас его еще нет, я неравнодушен к Sublime (**https://www.sublimetext.com/**)). Когда вы откроете его, скопируйте и вставьте следующий код в файл `plot_demo.py`:

```
import time

import numpy as np
import streamlit as st

progress_bar = st.sidebar.progress(0)
status_text = st.sidebar.empty()
last_rows = np.random.randn(1, 1)
chart = st.line_chart(last_rows)

for i in range(1, 101):
    new_rows = last_rows[-1, :] + np.random.randn(5, 1).cumsum(axis=0)
    status_text.text("%i%% Complete" % i)
    chart.add_rows(new_rows)
    progress_bar.progress(i)
    last_rows = new_rows
    time.sleep(0.05)

progress_bar.empty()

# Виджеты Streamlit автоматически запускают скрипт сверху вниз. Поскольку
# эта кнопка не связана ни с какой другой логикой, она просто вызывает
```

```
# простой повтор.
st.button("Re-run")
```

Этот код в режиме реального времени генерирует данные, используя случайные числа, выбранные из нормального распределения со средним значением 0 и дисперсией 1, выполняет с ними некоторые арифметические операции и строит на основе полученных результатов линейный график.

К концу этой книги вы сможете очень быстро создавать подобные приложения. А пока давайте запустим это приложение локально, введя в нашем терминале следующий код:

```
streamlit run plot_demo.py
```

Этот программный код откроет новую вкладку с вашим приложением в веб-браузере по умолчанию. Мы увидим, как наше приложение работает, это показано на рисунке ниже.

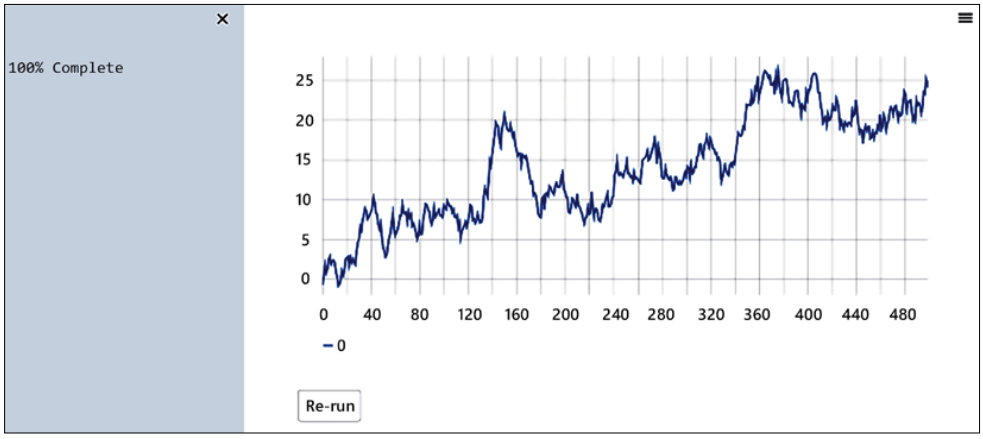


Рис. 1.1 Демонстрация построения графика

Вот именно так мы будем запускать каждое приложение Streamlit, сначала вызывая `streamlit run`, а затем указывая Streamlit скрипт Python, в котором находится код нашего приложения. Теперь давайте кое-что изменим в приложении, чтобы лучше понять, как работает Streamlit. Следующий код изменяет количество случайных чисел, которые мы отображаем на нашем графике, но не стесняйтесь вносить любые нужные изменения. Внесите изменения, используя нижеприведенный код, сохраните их в выбранном вами текстовом редакторе и снова запустите скрипт.

```
import time

import numpy as np
import streamlit as st

progress_bar = st.sidebar.progress(0)
```

```
status_text = st.sidebar.empty()
last_rows = np.random.randn(1, 1)
chart = st.line_chart(last_rows)

for i in range(1, 101):
    new_rows = last_rows[-1, :] + np.random.randn(50, 1).cumsum(axis=0)
    status_text.text("%i%% Complete" % i)
    chart.add_rows(new_rows)
    progress_bar.progress(i)
    last_rows = new_rows
    time.sleep(0.05)

progress_bar.empty()

# Виджеты Streamlit автоматически запускают скрипт сверху вниз. Поскольку
# эта кнопка не связана ни с какой другой логикой, она просто вызывает
# простой повтор.
st.button("Re-run")
```

Заметьте, что Streamlit обнаружил изменение в исходном файле и предлагает вам повторно запустить файл. Нажмите **Rerun** (или **Always rerun**, если хотите, чтобы это поведение использовалось по умолчанию; я, например, почти всегда так делаю) и наблюдайте, как меняется ваше приложение.

Не стесняйтесь попробовать внести некоторые другие изменения в приложение для построения графиков, чтобы освоиться! Когда набьете руку, можно перейти к созданию собственных приложений.

Создание приложения с нуля

Теперь, когда мы протестировали приложения, созданные другими, давайте создадим свое собственное!

Во-первых, давайте удостоверимся, что мы находимся в правильном каталоге (ранее мы называли его `streamlit_apps`), создадим новую папку с именем `clt_app` и в ней – новый файл.

Следующий код создает новую папку под названием `clt_app` и снова создает пустой Python'овский файл под названием `clt_demo.py`:

```
mkdir clt_app
cd clt_app
touch clt_demo.py
```

Всякий раз, когда мы запускаем новое приложение Streamlit, нужно обязательно импортировать Streamlit (в этой книге и других источниках часто используется алиас `st`). Streamlit предлагает уникальные функции для каждого типа контента (текст, графики, изображения и другие медиафайлы), которые мы можем использовать в качестве строительных блоков для всех наших приложений. Первая функция, которую мы будем использовать, – это `st.write()`. Она является функцией, которая принимает строку (и, как

мы увидим позже, почти любые объекты Python, например словари) и записывает ее непосредственно в наше веб-приложение во время вызова. Поскольку мы вызываем Python'овский скрипт, Streamlit последовательно просматривает файл и каждый раз, когда она видит одну из функций, назначает последовательный слот для этого фрагмента содержимого. Это делает платформу очень простой в использовании. Вы можете написать на Python все, что захотите. Когда нужно, чтобы что-то появилось в вашем приложении, вы можете просто использовать `st.write()`, и все будет готово.

В нашем файле `clt_demo.py` мы можем начать с основного вывода `'Hello World'`, используя `st.write()`. Для этого пишем следующий код:

```
import streamlit as st
st.write('Hello World')
```

Теперь можно проверить наш скрипт, запустив в терминале следующий программный код:

```
streamlit run clt_demo.py
```

Мы должны увидеть строку `'Hello World'`, напечатанную в нашем приложении. На следующем рисунке показан скриншот нашего приложения в Safari.

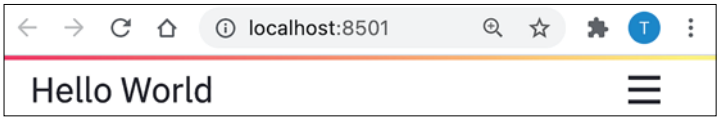


Рис. 1.2 Приложение Hello World

На рис. 1.2 следует отметить три элемента. Во-первых, мы видим строку такой, какой мы ее написали, и это здорово. Далее мы видим, что URL-адрес указывает на **localhost:8501**, это просто говорит нам, что мы размещаем приложение локально (т. е. его нет в интернете) через порт 8501. Нам почти ничего не нужно знать о системе портов на компьютерах или **протоколе передачи данных (Transmission Control Protocol – TCP)**. Здесь важно то, что это приложение является локальным для вашего компьютера. Третий важный элемент, на который следует обратить внимание, – это значок с тремя горизонтальными чертами ☰ (или с тремя точками ⋮ в более поздних версиях) в правом верхнем углу. На рис. 1.3 показано, что происходит, когда мы нажимаем значок.

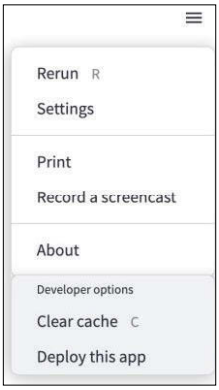


Рис. 1.3 Параметры

Перед нами панель параметров по умолчанию для приложений Streamlit. В этой книге мы подробно обсудим каждую из этих опций, особенно те, которые не

говорят сами за себя, такие как **Clear cache** (Очистить кеш). Все, что нам нужно знать на данный момент, – это то, что, если мы хотим повторно запустить приложение или найти настройки или документацию, мы можем воспользоваться этим значком.

Когда мы размещаем приложения, чтобы другие могли их использовать, они будут видеть тот же значок, при этом у них будут немного другие настройки параметров (например, они не смогут очистить кеш). Мы обсудим это более подробно позже.

Теперь сгенерируем распределение, из которого будем осуществлять отбор с возвращением. Я выбрал биномиальное распределение. Следующий программный код отвечает за симуляцию 1000 бросков монеты с помощью питоновского пакета numpy и печать среднего количества выпавших «орлов» из этих 1000 бросков.

```
import streamlit as st
import numpy as np
binom_dist = np.random.binomial(1, .5, 100)
st.write(np.mean(binom_dist))
```

Теперь, учитывая то, что мы знаем о центральной предельной теореме, мы могли бы ожидать, что, если формировать выборку из `binom_dist` достаточное количество раз, распределение средних значений этих выборок будет аппроксимировать нормальное распределение.

Мы уже обсуждали функцию `st.write()`. Теперь воспользуемся функцией `st.pyplot()`, которая позволяет нам применить все возможности популярной библиотеки matplotlib и перенести наш график matplotlib в Streamlit. Как только мы создадим изображение в matplotlib, можно явно указать Streamlit записать его в наше приложение с помощью функции `st.pyplot()`.

Итак, теперь собираем все вместе! Наше приложение имитирует 1000 подбрасываний монет и сохраняет результаты, подчиняющиеся биномиальному распределению, в список `binom_dist`. Затем отбираем (с возвращением) 100 значений из этого списка, берем среднее значение и сохраняем его в список `list_of_means`. Мы проделываем это 1000 раз (это перебор, можно было обойтись несколькими десятками выборок), а затем строим гистограмму. После того как мы это сделаем, результатом должно стать распределение в форме колокола.

Создаем пустой файл `clt_demo2.py`, введя в терминале

```
touch clt_demo2.py
```

и сохраняем в него следующий программный код:

```
import streamlit as st
import numpy as np
import matplotlib.pyplot as plt

binom_dist = np.random.binomial(1, .5, 1000)
```

```
list_of_means = []
for i in range(0, 1000):
    list_of_means.append(
        np.random.choice(binom_dist, 100, replace=True).mean())

fig, ax = plt.subplots()
ax = plt.hist(list_of_means)
st.pyplot(fig)
```

Останавливаем приложение, ранее запущенное в терминале, с помощью сочетания клавиш **Ctrl + C** и запускаем в терминале наше новое приложение:

```
streamlit run clt_demo2.py
```

Каждый запуск этого приложения будет создавать новую колоколообразную гистограмму. Когда я запустил его, моя гистограммв выглядела, как на рис. 1.4. Если ваш график отличается от того, что вы видите на следующем рисунке, это совершенно нормально из-за случайной выборки, используемой в нашем коде.

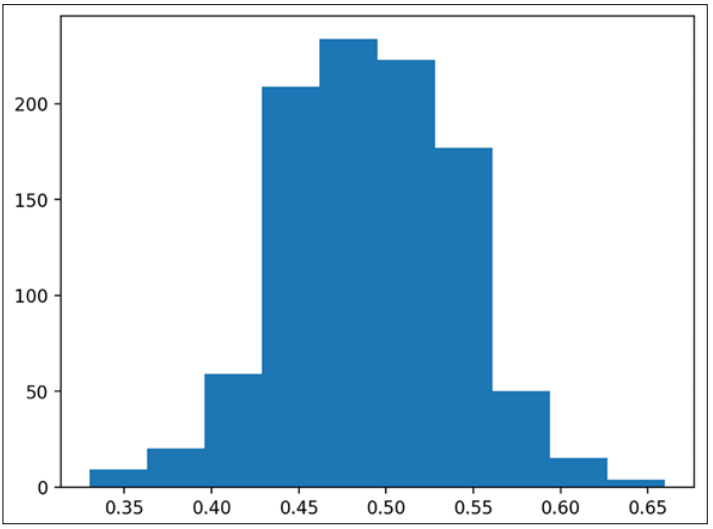


Рис. 1.4. Колоколообразная гистограмма

Как вы, наверное, заметили, сначала мы создали пустое изображение и пустые области рисования для этого изображения, вызвав `plt.subplots()`, а затем записываем созданную нами гистограмму в переменную `ax`. Благодаря этому мы явно указываем Streamlit показать рисунок в нашем приложении.

Это важный шаг, поскольку в Streamlit мы можем пропустить этот шаг и не записывать нашу гистограмму в какую-либо переменную, чтобы потом сразу вызвать `st.pyplot()`. Создаем пустой файл `clt_demo3.py`, вставляем в него программный код, который демонстрирует новый подход:

```
import streamlit as st
import numpy as np
import matplotlib.pyplot as plt

binom_dist = np.random.binomial(1, .5, 1000)
list_of_means = []
for i in range(0, 1000):
    list_of_means.append(
        np.random.choice(binom_dist, 100, replace=True).mean())

plt.hist(list_of_means)
st.pyplot()
```

Запускаем в терминале наше приложение:

```
streamlit run clt_demo3.py
```

Я не рекомендую этот метод, так как он может дать неожиданные результаты. Возьмем пример, в котором мы хотим сначала создать нашу гистограмму средних, а затем – еще одну гистограмму из нового списка, заполненного единицами.

Для этого создаем пустой файл `clt_demo4.py`, введя в терминале

```
touch clt_demo4.py
```

и сохраняем в него следующий программный код:

```
import streamlit as st
import numpy as np
import matplotlib.pyplot as plt

binom_dist = np.random.binomial(1, .5, 1000)
list_of_means = []
for i in range(0, 1000):
    list_of_means.append(
        np.random.choice(binom_dist, 100, replace=True).mean())

plt.hist(list_of_means)
st.pyplot()
plt.hist([1,1,1,1])
st.pyplot()
```

Сколько графиков мы получим? Каким будет результат?

```
streamlit run clt_demo4.py
```

Я ожидаю, что он покажет две гистограммы: первую, созданную на основе списка `list_of_means`, и вторую на основе списка из единиц.

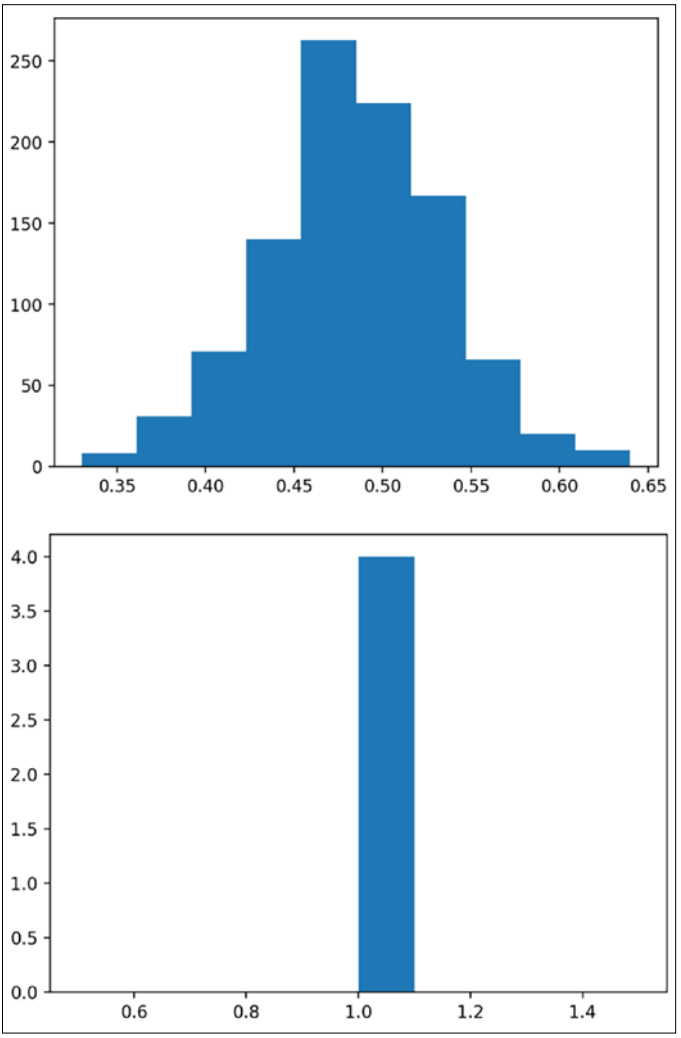


Рис. 1.5. Сказка о двух гистограммах

Все хорошо, но лучше явно создать два графика. Если мы явно создадим два графика, можем вызвать функцию `st.pyplot()` в любом месте после того, как график был сгенерирован, и проконтролировать размещение наших графиков.

Создаем пустой файл `clt_demo5.py`, введя в терминале

```
touch clt_demo5.py
```

и сохраняем в него следующий программный код, который явно разделяет два графика:

```
import streamlit as st
import numpy as np
```

```
import matplotlib.pyplot as plt

binom_dist = np.random.binomial(1, .5, 1000)
list_of_means = []
for i in range(0, 1000):
    list_of_means.append(
        np.random.choice(binom_dist, 100, replace=True).mean())

fig1, ax1 = plt.subplots()
ax1 = plt.hist(list_of_means)
st.pyplot(fig1)

fig2, ax2 = plt.subplots()
ax2 = plt.hist([1,1,1,1])
st.pyplot(fig2)
```

Давайте убедимся.

```
streamlit run clt_demo5.py
```

Предыдущий код строит обе гистограммы отдельно, сначала определяя отдельные переменные для каждого изображения и областей рисования с помощью функции `plt.subplots()`, а затем записывая гистограмму в соответствующую область рисования. После этого мы можем вызвать `st.pyplot()`, используя созданное изображение, и получаем следующее приложение (рис. 1.6).

Мы можем ясно увидеть на рис. 1.6, что две гистограммы теперь разделены, что является желаемым поведением. Мы очень часто будем создавать несколько визуализаций в Streamlit и использовать именно этот метод. Теперь поговорим о пользовательском вводе!

Использование пользовательского ввода в приложениях Streamlit

На данный момент наше приложение – это просто причудливый способ показать наши визуализации. Но большинство веб-приложений требуют некоторого пользовательского ввода или являются динамическими, а не статическими визуализациями. К счастью для нас, Streamlit предлагает множество функций для приема входных данных от пользователей, и все они различаются по объекту, который мы хотим ввести. Есть поля для ввода текстовых данных произвольной формы, которые задаются с помощью функции `st.text_input()`; есть переключатели, задающиеся с помощью функции `st.radio()`; есть поля для ввода количественных данных, они задаются с помощью `st.number_input()`; и еще дюжина чрезвычайно полезных функций для создания приложений Streamlit. В этой книге мы подробно рассмотрим большинство из них, но начнем с ввода количественных данных.

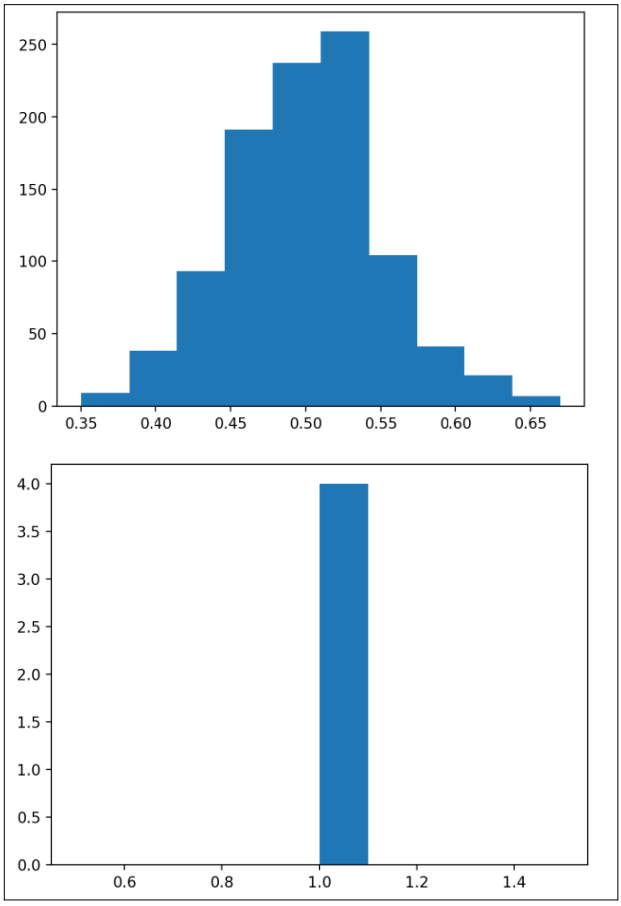


Рис. 1.6. Гистограммы, зафиксированные в отдельных переменных

В предыдущем примере мы предположили, что монеты, которые мы подбрасывали, были честными монетами и с вероятностью 50/50 выпадали «орлом» или «решкой». Давайте позволим пользователю самому определять вероятность выпадения «орла» в процентах, запишем ее в переменную и будем использовать ее в качестве входных данных в нашем биномиальном распределении. В функцию ввода количественных данных передаем надпись, минимальное и максимальное значение, а также значение по умолчанию.

Создаем пустой файл `clt_demo6.py`, введя в терминале

```
touch clt_demo64.py
```

и сохраняем в него соответствующий программный код:

```
import streamlit as st
import numpy as np
```

```
import matplotlib.pyplot as plt

perc_heads = st.number_input(
    label="Вероятность выпадения 'орла'",
    min_value=0.0, max_value=1.0, value=.5
)

binom_dist = np.random.binomial(1, perc_heads, 1000)

list_of_means = []
for i in range(0, 1000):
    list_of_means.append(
        np.random.choice(binom_dist, 100, replace=True).mean())

fig, ax = plt.subplots()
ax = plt.hist(list_of_means, range=[0,1])
st.pyplot(fig)
```

Запускаем:

```
streamlit run clt_demo6.py
```

Предыдущий программный код использует функцию `st.number_input()` для фиксации вероятности выпадения «орла», записывает пользовательский ввод в переменную (`perc_heads`), затем использует ее для изменения вводимых данных в нашей функции биномиального распределения, которую мы использовали ранее. Он также устанавливает значения оси X нашей гистограммы в диапазоне между 0 и 1, чтобы мы могли лучше заметить изменения при изменении ввода. Попробуйте немного поиграть с этим приложением: измените вероятность выпадения «орла» и обратите внимание, как приложение реагирует на изменение пользовательского ввода. На рис. 1.7 показан результат, когда мы задали вероятность выпадения «орла» равной 0,25.

Кроме того, в Streamlit мы можем принимать ввод текстовых данных с помощью функции `st.text_input()`, точно так же как мы принимали ввод количественных данных.

Создаем пустой файл `clt_demo7.py`, введя в терминале

```
touch clt_demo7.py
```

и сохраняем в него программный код, который принимает ввод текстовых данных и создает на их основе заголовок:

```
import streamlit as st
import numpy as np
import matplotlib.pyplot as plt

perc_heads = st.number_input(
    label="Вероятность выпадения 'орла'",
```

```
    min_value=0.0, max_value=1.0, value=.5
)

graph_title = st.text_input(label="Заголовок графика")

binom_dist = np.random.binomial(1, perc_heads, 1000)

list_of_means = []
for i in range(0, 1000):
    list_of_means.append(
        np.random.choice(binom_dist, 100, replace=True).mean())

fig, ax = plt.subplots()
plt.hist(list_of_means, range=[0,1])
plt.title(graph_title)
st.pyplot(fig)
```

Запускаем:

```
streamlit run clt_demo7.py
```

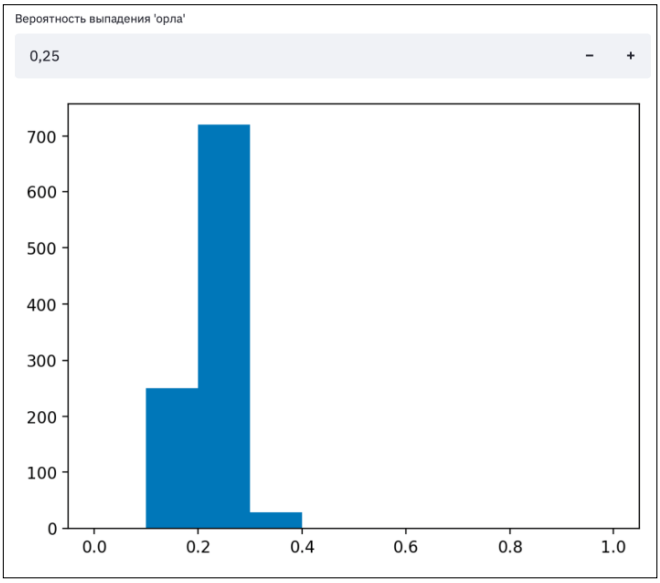


Рис. 1.7. Результат, который мы получили при вероятности выпадения «орла» 0,25

Этот программный код создает приложение Streamlit с двумя полями, у нас будет поле для ввода количественных данных и поле для ввода текстовых данных, оба поля можно использовать для изменения нашего приложения Streamlit. В итоге получаем приложение Streamlit, которое выглядит, как на рис. 1.8, с динамически изменяемыми заголовком и вероятностью выпадения «орла».