Оглавление

Об авторе

Элиас Даббас – практикующий специалист по онлайн-маркетингу, а также обработке и анализу данных. Совместив эти области, он нашел себя в проектировании программного обеспечения с открытым кодом для создания дашбордов и приложений для работы с данными. Также он специализируется на создании программ для интернет-маркетинга.

Элиас является автором популярной библиотеки *advertools* для Python, предлагающей богатый выбор маркетинговых инструментов с уклоном в оптимизацию поисковых систем (SEO), поисковый маркетинг (SEM), сбор данных и текстовый анализ.

О рецензенте

Леонардо Феррейра – бухгалтер, самостоятельно освоивший обработку и анализ данных до уровня Kaggle Grandmaster и выступающий разработчиком платформ в области аналитики данных. Он начал свое обучение в 2017 году и уже через несколько месяцев приступил к работе по изучаемой специальности. С тех пор Леонардо успел поработать в крупных бразильских и международных компаниях, реализовав более сотни проектов с открытым исходным кодом с портфолио на GitHub и Kaggle. Обладает статусом *Top Rated Plus* на фриланссерской платформе *Upwork*, в рамках которой реализовал более 20 проектов по анализу данных. Также интересуется решениями на базе блокчейн-платформы Cardano.

Введение

Фреймворк Dash от Plotly позволяет разработчикам Python создавать полноценные приложения для аналитической работы с данными и интерактивные дашборды. Книга, которую вы держите в руках, призвана помочь вам исследовать богатый функционал фреймворка Dash по визуализации данных и научиться извлекать максимум возможного из исходной информации.

Начнем мы с описания экосистемы Dash, основных пакетов, входящих в состав этого фреймворка, а также сторонних библиотек, позволяющих структурировать данные для вашего приложения.

После этого приступим к созданию первого приложения с использованием фреймворка Dash и добавлению в него базового функционала. Далее вы познакомитесь с такими специфическими элементами приложений, как выпадающий список, флажок, ползунок, календарь и др., а также научитесь связывать их с диаграммами и прочими элементами вывода. В зависимости от данных, которые вы визуализируете, вы будете использовать наиболее подходящие типы диаграмм, включая точечные диаграммы, линейные графики, столбчатые диаграммы, гистограммы, карты и пр., и узнаете, как можно адаптировать их под собственные нужды.

Прочитав эту книгу, вы сможете разрабатывать и развертывать сложные интерактивные дашборды, производить многоступенчатый рефакторинг кода и оптимизировать написанные вами приложения.

Для кого эта книга

Книга, которую вы начинаете читать, предназначена для специалистов по работе с данными и аналитиков, желающих больше узнать о своих исходных данных при помощи интерактивных дашбордов, включающих полный спектр визуализаций. Предполагается, что вы хотя бы на базовом уровне знаете язык программирования Python. Это поможет вам быстрее и лучше усвоить техники, описанные в книге.

Структура книги

Глава 1. Знакомство с экосистемой Dash. В данной главе вы познакомитесь с общей экосистемой фреймворка Dash, пакетами, входящими в его состав, а также сторонними библиотеками. Прочитав эту вводную главу, вы научитесь отличать разные элементы приложения и узнаете, для чего предназначен каждый из них. В качестве бонуса вы даже создадите свое первое простое приложение.

Глава 2. Структура приложений Dash. Из этой главы вы узнаете, как можно добавить созданному ранее приложению интерактивности. Здесь вы познакомитесь с концепцией *обратных вызовов* (callback) и научитесь объединять разные визуальные элементы приложения. Вы также увидите, как

с помощью функций обратного вызова можно позволить пользователю управлять содержимым одного визуального элемента посредством другого.

Глава 3. Работа с объектом Figure. В третьей главе книги вы познакомитесь с ключевым объектом *Figure*, узнаете о его компонентах, а также о способах управления им и преобразовании его в различные форматы. Позже мы используем полученные навыки для создания особых типов диаграмм для нашего приложения.

Глава 4. Подготовка и преобразование данных. Введение в Plotly Express. Здесь вы узнаете о форматах данных, наиболее пригодных для анализа. Также вы познакомитесь с пакетом Plotly Express и увидите, с какой легкостью можно с его помощью создавать диаграммы и связывать данные с элементами визуализации.

Глава 5. Интерактивное сравнение данных при помощи столбчатых диаграмм и выпадающих списков. В этой главе мы немного глубже погрузимся в функционал диаграмм и изучим дополнительные возможности визуализации данных. После этого вы узнаете, как можно дать пользователю возможность выбрать сравниваемые элементы при помощи выпадающих списков.

Глава 6. Исследование переменных при помощи точечной диаграммы и фильтрация наборов данных. В данной главе мы подробно рассмотрим один из наиболее популярных видов визуализации, а именно диаграмму рассеяния или точечную диаграмму. Как и в случае со столбчатой диаграммой, мы рассмотрим различные варианты настройки этого типа визуализации. Точечные диаграммы отличаются очень богатыми возможностями для настройки, включая управление размером точек в зависимости от выбранной переменной, исключение наложения точек друг на друга и вывод диаграммы с большим количеством точек данных.

Глава 7. Работа с географическими картами и обогащение дашбордов при помощи языка разметки Markdown. В этой главе вы познакомитесь с еще одним распространенным типом визуализации. Существует множество способов отображения данных на карте. Мы рассмотрим два наиболее часто используемых: *точечный* (scatter map) и *картограмма* (choropleth map).

Глава 8. Определение частотности данных с помощью гистограмм и построение интерактивных таблиц. Эта глава посвящена разным способам построения гистограмм и их настройки, а также разделению данных различными методами с последующим подсчетом результирующих значений.

Глава 9. Машинное обучение: пусть данные говорят сами за себя. В этой главе вы узнаете о том, как работает кластеризация данных, и научитесь оценивать качество анализа. Также мы рассмотрим технику оценки кластеров и даже разработаем интерактивное приложение с реализацией кластеризации по методу *k*-средних.

Глава 10. Ускорение работы приложений с помощью улучшений функций обратного вызова. Здесь мы поговорим об использовании обратных вызовов на базе шаблонов с целью динамической модификации приложения на основе взаимодействия с пользователем и других факторов.

Глава 11. Ссылки и многостраничные приложения. В данной главе будет представлена новая архитектура, позволяющая создавать многостраничные приложения. Также мы рассмотрим технику использования ссылок в качестве элементов ввода или вывода со взаимодействием с другими элементами приложения.

Глава 12. Развертывание приложения. В этой главе мы обсудим вопросы развертывания созданного приложения на сервере с возможностью доступа к нему пользователям из любой точки мира. Здесь возможны разные варианты, и мы рассмотрим пару простых опций, которые могут оказаться полезными.

Глава 13. Следующие шаги. В заключительной главе книги мы поговорим о том, как можно вывести написанное приложение на новый уровень. Здесь мы дадим определенные рекомендации, советы и ресурсы, которые вам, возможно, захочется изучить самостоятельно.

Как извлечь максимум из книги

Для выполнения примеров из книги вам понадобится стабильное соединение с интернетом.

Если вы читаете книгу в формате PDF, мы рекомендуем вводить программный код вручную или использовать загруженный код из хранилища на GitHub (ссылка будет указана ниже). Это позволит вам избежать ошибок, связанных с копированием и вставкой текста.

Загрузите сопроводительные файлы

Сопроводительные файлы можно загрузить на странице книги на сайте издательства **www.dmkpress.com**.

Загрузите цветные изображения

По следующей ссылке вы можете скачать в виде PDF все рисунки и диаграммы, использованные в книге: **https://static.packt-cdn.com/downloads/9781800568914_ColorImages.pdf**.

Книга в видеофрагментах

Сопроводительные видеофрагменты к этой книге можно посмотреть по адресу **https://bit.ly/3vaXYQJ**.

Условные обозначения

На протяжении книги мы будем использовать следующие условные обозначения и шрифты.

`Код в тексте`: так в тексте книги мы будем обозначать код, имена таблиц баз данных, имена папок, файлов, расширения файлов, пути, ссылки, пользовательский ввод. Пример: «Наш набор данных будет состоять из файлов в папке `data`, находящейся в корне репозитория».

Блоки кода будут выделены следующим образом:

```
import plotly.express as px
gapminder = px.data.gapminder()
gapminder
```

Важные места в коде будут подсвечены жирным шрифтом, как показано ниже:

```
import os
import pandas as pd
pd.options.display.max_columns = None
os.listdir('data')
['PovStats Series .csv',
 'PovStats Country .csv',
 'PovStats Country-Series .csv',
 'PovStats Data .csv',
 'PovStats FootNote .csv']
```

Жирным шрифтом также будут выделяться новые термины, важные слова и текст, который вы видите на экране. Например, таким образом будут обозначаться пункты меню. Пример:

«Еще одним важным столбцом является столбец **Limitations and exceptions**».

> **Советы или важные примечания**
>
> Будут выводиться так.

Часть I

Построение приложения на Dash

В этой вводной части вы познакомитесь с экосистемой фреймворка Dash и напишете свое первое простое приложение с минимальным функционалом.

Содержание этой части:

- глава 1 *«Знакомство с экосистемой Dash»;*
- глава 2 *«Структура приложений Dash»;*
- глава 3 *«Работа с объектом Figure»;*
- глава 4 *«Подготовка и преобразование данных. Введение в Plotly Express».*

Глава 1

Знакомство с экосистемой Dash

При работе с данными происходят постоянные изменения в объеме анализируемых данных, их источниках и типах. В связи с этим очень важно иметь возможность легко и просто комбинировать любые объемы данных из различных источников. Фреймворк *Dash* – это не только про исследование данных. Это про почти все стадии процесса анализа данных: от их поиска до создания полноценной рабочей среды.

В этой вводной главе мы познакомимся с экосистемой Dash и сконцентрируемся на внешнем макете приложения – той его части, с которой взаимодействует пользователь. Прочитав эту главу, вы сможете создать полностью работающее приложение с любыми визуальными элементами, но без интерактивных возможностей.

Темы, которые будут рассмотрены в главе:

- настройка окружения;
- исследование фреймворка Dash и сопутствующих пакетов;
- введение в базовую структуру приложения Dash;
- создание и запуск простого приложения Dash;
- добавление HTML и других компонентов в приложение;
- проектирование макета и управление темами.

Технические требования

В каждой главе будут применяться свои технические требования, но некоторые из них будут актуальны на протяжении всей книги.

Прежде всего у вас должен быть установлен *Python* 3.6 или выше, который можно загрузить по адресу **https://www.python.org**. Также вам понадобится текстовый редактор или любая *интегрированная среда разработки* (integrated development environment – IDE) для написания и редактирования кода.

В этой главе мы будем использовать пакеты Dash, Dash HTML Components и Dash Bootstrap Components, которые можно загрузить вместе с остальными пакетами, следуя инструкции из следующего раздела. Исходный код и данные для этой книги можно скачать в репозитории GitHub по адресу **https://github.com/**

PacktPublishing/Interactive-Dashboards-and-Data-Apps-with-Plotly-and-Dash. Как я уже упомянул, в следующем разделе мы детально остановимся на настройке вашего рабочего окружения.

Исходный код к этой главе располагается в хранилище GitHub по адресу **https://github.com/PacktPublishing/Interactive-Dashboards-and-Data-Apps-with-Plotly-and-Dash/tree/master/chapter_01**.

Сопроводительные видеофрагменты к этой главе можно посмотреть по адресу **https://bit.ly/3atXPjc**.

Настройка окружения

Поскольку все пакеты, используемые в этой книге, стремительно развиваются и меняются, вы можете столкнуться с различиями в поведении ваших приложений. Чтобы в точности воспроизвести функционал приложений, заложенный при написании этой книги, мы рекомендуем вам клонировать репозиторий книги, установить версии пакетов, которые использовались при написании книги, и использовать в своих примерах приведенные наборы данных. Откройте командную строку, перейдите в папку, в которой хотите создать проект, и выполните следующие действия.

1. Создайте виртуальное окружение Python в папке с именем `dash_project` (или любой другой на ваше усмотрение). Это также приведет к созданию папки с выбранным именем:

```
python3 -m venv dash_project
```

2. Активируйте виртуальное окружение.

   В Unix или macOS выполните следующую инструкцию:

```
source dash_project/bin/activate
```

   Для Windows инструкция будет такой:

```
dash_project\Scripts\activate.bat
```

3. Перейдите в созданную папку:

```
cd dash_project
```

4. Клонируйте репозиторий книги на GitHub:

```
git clone https://github.com/PacktPublishing/Interactive-
Dashboards-and-Data-Apps-with-Plotly-and-Dash
```

5. В вашей папке должен появиться файл `requirements.txt` с перечислением всех необходимых пакетов и их версий. Вы можете установить все эти пакеты, перейдя в папку репозитория и выполнив команду `install`, как показано ниже:

```
cd Interactive-Dashboards-and-Data-Apps-with-Plotly-and-Dash/
pip install -r requirements.txt
```

В папке `data` вы обнаружите копии наборов данных, которые были загружены с сайта **https://datacatalog.worldbank.org/dataset/poverty-and-equity-database**. Вы всегда можете загрузить свежие версии файлов, но здесь как с версиями пакетов – лучше использовать в точности те данные, которые применялись при написании книги, чтобы получать такие же результаты.

Для корректного отображения объектов и приложений Plotly в JupyterLab необходимо также установить Node.js по адресу **https://nodejs.org**.

Еще вам нужно установить расширение *JupyterLab Plotly*, что можно сделать, запустив следующую инструкцию из командной строки в вашем виртуальном окружении:

```
jupyter labextension install jupyterlab-plotly@4.14.1
```

Обратите внимание, что номер версии в конце строки должен совпадать с версией Plotly, которую вы используете. Вы можете изменить версию, не забыв при этом обновить и сам пакет Plotly.

Теперь вы полностью готовы двигаться дальше. В каждой следующей главе мы будем развивать идеи, озвученные в предыдущих главах, и дорабатывать созданные приложения, улучшая их функционал.

Основная цель – дать вам как можно больше возможностей для практики. Создать отдельный компонент Dash не составляет труда, но при объединении нескольких компонентов в приложении могут начаться сложности. Вы прочувствуете это, когда вам придется обновлять макет приложения и выполнять рефакторинг кода, концентрируясь на деталях, но в то же время не упуская из вида картину в целом.

Итак, окружение мы подготовили, пришло время рассказать о фреймворке Dash.

Исследование фреймворка Dash и сопутствующих пакетов

Хотя в этом нет строгой необходимости, все же вам полезно будет узнать, какие основные компоненты используются в *Dash*, чтобы уверенно чувствовать себя при разработке более сложных приложений и знать, куда обращаться за помощью.

Рис. 1.1. Из чего сделан Dash

Как видно на рис. 1.1, Dash использует фреймворк *Flask* на стороне сервера. Для построения диаграмм применяется графическая библиотека *Plotly* – это не строгое требование, но эта библиотека обладает самыми богатыми возможностями и поддержкой. Библиотека *React* используется для управления компонентами. По сути, любое приложение Dash можно воспринимать как одно-

страничное приложение React. Но гораздо важнее для нас сейчас узнать, какие пакеты используются в процессе создания приложения, и именно об этом мы поговорим далее.

> **Примечание**
>
> Одним из главных преимуществ фреймворка Dash является то, что он позволяет создавать полностью интерактивные приложения и интерфейсы для работы с данными и аналитикой с использованием чистого Python и без необходимости изучать HTML, CSS или JavaScript.

> **Совет**
>
> Тем, кто знаком с библиотекой Matplotlib, будет полезно узнать, что существуют инструменты для преобразования объектов Matplotlib в объекты Plotly. Таким образом, создав элемент в Matplotlib, вы сможете сконвертировать его в Plotly с помощью всего одной функции `mpl_to_plotly`. На момент написания книги такой функционал поддерживался только для Matplotlib версии не выше 3.0.3. Ниже приведен пример использования этой функции.

```
%config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt
from plotly.tools import mpl_to_plotly

mpl_fig, ax = plt.subplots()
ax.scatter(x=[1, 2, 3], y=[23, 12, 34])
plotly_fig = mpl_to_plotly(mpl_fig)
plotly_fig
```

Пакеты, содержащиеся во фреймворке Dash

Dash – это не один большой пакет, который содержит все необходимое. Это, скорее, собрание пакетов, каждый из которых служит конкретной цели. В дополнение, как мы увидим позже, существует большое количество сторонних пакетов, которые используются совместно с Dash, а сообщество создает собственные библиотеки для работы с этим фреймворком.

Ниже приведены основные пакеты, входящие в состав фреймворка Dash, которые мы будем изучать в этой главе:

- **Dash:** это базовый пакет, представляющий основу любого приложения посредством объекта `dash.Dash`. Также в этом пакете представлен функционал для управления интерактивностью и исключениями, что вы увидите при создании приложения;

- **Dash Core Components:** этот пакет содержит интерактивные компоненты, которыми управляет пользователь. Выпадающие списки, календари, ползунки и многое другое – это все есть в этом пакете. В главе 2 мы будем подробно говорить об этих компонентах применительно к интерактивности приложений, а во второй части книги еще более детально обсудим нюансы их использования;
- **Dash HTML Components:** в этом пакете представлены все возможные теги HTML в виде классов Python.Именно здесь происходит преобразование Python в HTML. К примеру, вы можете написать в Python `dash_html_components.H1('Hello, World')`, и этот код сгенерирует следующую разметку HTML: `<h1>Hello, World</h1>` – и соответствующим образом отобразит ее в браузере;
- **Dash Bootstrap Components:** это сторонний пакет, добавляющий фреймворку Dash функциональности Bootstrap. Этот пакет и его компоненты отвечают главным образом за макет и визуальное отображение элементов. Используя его, можно, например, разместить элементы приложения бок о бок или один над другим, определяя их размеры в зависимости от размера окна браузера, или настроить цветовую гамму приложения особым образом.

> **Совет**
>
> Чтобы установить все основные пакеты Dash, достаточно установить главный пакет. Это также позволит сохранить консистентность версий сопутствующих пакетов. Просто выполните инструкцию `pip install dash` из командной строки. Для обновления пакетов воспользуйтесь командой `pip install dash --upgrade`.

Теперь пришло время взглянуть на базовую структуру типичного приложения Dash.

Введение в базовую структуру приложения Dash

На рис. 1.2 условно показан процесс создания приложения Dash. К примеру, у нас есть файл с именем `app.py` (вы можете назвать его по своему усмотрению). Содержимое файла показано в правой колонке рисунка с условным разделением на секции, а в левой приведено описание секции.

Давайте рассмотрим каждый из приведенных этапов отдельно:

- **импорт (стандартная заготовка)**: как и любой модуль Python, мы начинаем написание приложения с импортирования необходимых пакетов, давая им привычные псевдонимы[1];

[1] Здесь и далее: начиная с версии фреймворка Dash 2.0 следует использовать следующий синтаксис импорта: `from dash import html` и `from dash import dcc`, поскольку указанный синтаксис является устаревшим. – *Прим. перев.*

Части приложения	app.py

Импорт (стандартная заготовка)	`import dash`
	`import dash_html_components as html`
	`import dash_core_components as dcc`

Создание экземпляра приложения	`app = dash.Dash(__name__)`

Макет приложения: список HTML и/или интерактивных компонентов	`app.layout = html.Div([`
	`  dcc.Dropdown()`
	`  dcc.Graph()`
	`  ...`
	`])`

Функции обратного вызова	`@app.callback()`
	`  ...`
	`@app.callback()`
	`  ...`

Запуск приложения	`if __name__ == '__main__':` `  app.run_server()`

Рис. 1.2. Структура приложения Dash

- **создание экземпляра приложения**: простой способ создать приложение посредством инициализации переменной `app`. В качестве параметра передается значение `__name__`, чтобы Dash мог легко находить статические ресурсы, используемые в приложении;
- **макет приложения**: этому этапу мы посвятим большую часть данной главы. Именно здесь мы создаем все пользовательские элементы. Для этого мы обычно определяем контейнер (`html.Div`), принимающий в качестве аргумента `children` список дочерних компонентов. Эти компоненты будут последовательно отображаться при запуске приложения – один под другим. В следующем разделе мы создадим простейшее приложение с минималистическим макетом;
- **функции обратного вызова**: эту тему мы подробно начнем обсуждать в главе 2, посвященной интерактивности приложений. Пока вам достаточно будет знать, что здесь определяются функции для связывания визуальных элементов, в результате чего мы получаем функционал приложения. Обычно функции являются независимыми, они не должны быть объявлены внутри контейнера, а их порядок в модуле не имеет значения;
- **запуск приложения**: здесь мы осуществляем так называемый запуск приложения, если применить идиому о запуске модулей Python в качестве скриптов.

Итак, как я и обещал, мы уже готовы к написанию своих первых строк!

Создание и запуск простого приложения Dash

Держа в уме структуру приложения, которую мы только что обсудили, за исключением функций обратного вызова, давайте попробуем построить простейшее приложение.

Создайте файл app.py и введите в него следующий код.

1. Импорт необходимых пакетов с псевдонимами:

```
import dash
import dash_html_components as html
```

2. Создание экземпляра приложения:

```
app = dash.Dash(__name__)
```

3. Создание макета приложения:

```
app.layout = html.Div([
    html.H1('Hello, World!')
])
```

4. Запуск приложения:

```
if __name__ == '__main__':
    app.run_server(debug=True)
```

Позвольте сделать пару замечаний перед запуском приложения. Я настоятельно рекомендую не пользоваться копированием и вставкой кода. Пишите вручную. Вы должны запоминать вводимые конструкции. Кроме того, в процессе написания кода вам будут показываться подсказки в среде выполнения, и очень важно обращать внимание на предлагаемые возможности для компонентов, классов или функций.

Макет нашего приложения содержит единственный элемент, переданный как список элементу html.Div в качестве параметра children. Этот элемент будет преобразован в тег H1.

Заметьте также, что я передал на вход методу app.run_server параметр debug со значением True. Это позволяет активировать инструменты отладки, помогающие при разработке приложения.

Итак, мы готовы запустить наше первое приложение Dash. Для этого из командной строки, находясь в папке с файлом app.py, выполните следующую инструкцию:

```
python app.py
```

Если третья версия Python не установлена в вашей системе по умолчанию, вам может потребоваться указать номер версии вручную, как показано ниже:

```
python3 app.py
```

Вы увидите вывод, показанный на рис. 1.3, из которого следует, что приложение запущено.

```
> python app.py
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
```

Рис. 1.3. Вывод командной строки после запуска приложения

Поздравляем! Ваше первое приложение Dash готово! Теперь, если вы введете в адресную строку браузера ссылку, указанную в выводе командной строки (**http://127.0.0.1:8050**), то увидите строку *Hello, World!* в виде заголовка первого уровня (H1). Кроме того, в выводе командной строки указано, что в данный момент запущено серверное приложение Flask с именем `app`, а также присутствует предупреждение о том, что это сервер для разработки, а не для выпуска приложения. О развертывании приложений мы будем говорить позже, сейчас же вам достаточно знать, что этого сервера нам вполне хватит для разработки и тестирования приложения. Вывод приложения показан на рис. 1.4.

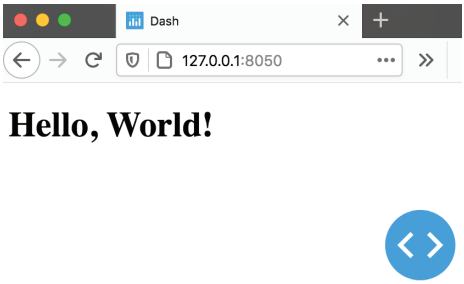

Рис. 1.4. Приложение в браузере

Под текстом, выведенным в виде заголовка первого уровня, вы можете заметить голубой кружок со стрелками. Щелкните по нему, и откроются дополнительные опции *отладки*, которые пригодятся вам при наличии в приложении функций обратного вызова и появлении ошибок. Если в качестве параметра при запуске приложения передать `debug=False`, эта кнопка отображаться не будет.

Теперь, когда вы создали простое приложение Dash и разобрались в базовой структуре макета, пришло время присмотреться к двум пакетам, которые используются для добавления и управления визуальными элементами. Первый пакет именуется **Dash HTML Components**, а второй – **Dash Bootstrap Components**.

Добавление HTML и других компонентов в приложение

С этого момента и до конца главы мы будем главным образом говорить об атрибуте нашего приложения `app.layout` и вносить в него изменения. Делать

это очень легко – достаточно просто добавлять элементы в список, поступающий на вход элемента `html.Div` в качестве параметра `children`:

```
html.Div(children=[component_1, component_2, component_3, ...])
```

Добавление компонентов HTML в приложение Dash

Поскольку доступные компоненты в пакете в точности соотносятся с соответствующими тегами HTML, этот пакет можно назвать достаточно устойчивым и стабильным. Давайте рассмотрим параметры, общие для всех его компонентов.

На момент написания книги **Dash HTML Components** включал в себя 131 компонент, общими для которых являются 20 параметров. Поговорим о самых популярных из них, которые мы часто будем использовать в этой книге:

- `children`: это основной (и первый) контейнер, вмещающий содержимое компонента. В нем может размещаться как один элемент, так и список элементов;
- `className`: то же, что и атрибут `class`, но под другим именем;
- `id`: несмотря на то что в данной главе мы не будем использовать этот параметр, он играет важнейшую роль при добавлении элементам интерактивности. В дальнейшем при построении приложения мы будем очень часто обращаться к этому свойству. Ну а пока вам достаточно знать, что вы можете присваивать своим компонентам произвольные идентификаторы, по которым впоследствии сможете обращаться к ним при настройке интерактивности;
- `style`: похож по назначению на одноименный атрибут HTML, но имеет некоторые отличия. Во-первых, атрибуты в нем задаются в так называемом верблюжьем стиле (camelCase). Допустим, вам нужно задать показанные ниже атрибуты:

```
<h1 style="color:blue; font-size: 40px; margin-left:
20%">A Blue Heading</h1>
```

Для этого вам придется определить их следующим образом:

```
html.H1(children='A Blue Heading',
    style={'color': 'blue',
        'fontSize': '40px',
        'marginLeft': '20%'})
```

Вы наверняка заметили, что значение атрибуту `style` присваивается с использованием словаря Python.

Другие параметры подчиняются собственным законам и правилам в зависимости от компонента, которому они принадлежат. Давайте потренируемся в добавлении элементов HTML в наше приложение. Вернемся к нашему исходному файлу `app.py`, поэкспериментируем с другими элементами HTML и

снова попробуем запустить приложение. Начало и конец приложения оставьте прежними, а изменить нужно главным образом `app.layout`, как показано ниже:

```
...
app = dash.Dash(__name__)
app.layout = html.Div([
    html.H1('Poverty And Equity Database',
        style={'color': 'blue',
            'fontSize': '40px'}),
    html.H2('The World Bank'),
    html.P('Key Facts:'),
    html.Ul([
        html.Li('Number of Economies: 170'),
        html.Li('Temporal Coverage: 1974 - 2019'),
        html.Li('Update Frequency: Quarterly'),
        html.Li('Last Updated: March 18, 2020'),
        html.Li([
            'Source: ',
            html.A('https://datacatalog.worldbank.org/dataset/poverty-and-
equity-database', href='https://datacatalog.worldbank.org/dataset/poverty-
and-equity-database')
        ])
    ])
])
...
```

В результате наше приложение приобретет вид, показанный на рис. 1.5.

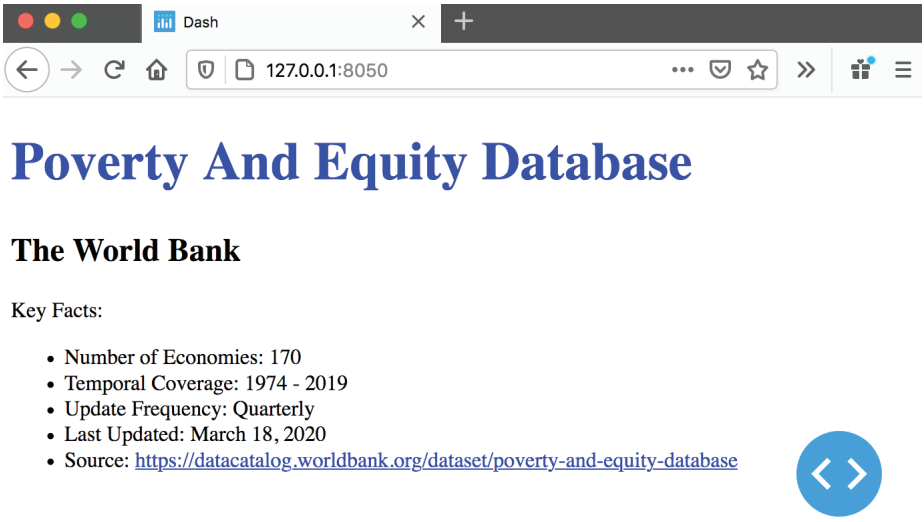

Рис. 1.5. Обновленное приложение, запущенное в браузере

> **Совет**
>
> Если вы знакомы с тегами языка разметки HTML, у вас не будет никаких проблем. Если нет, советую отдельно ознакомиться с этим языком. Начать можно на сайте W3Schools по адресу **https://www.w3schools.com/html**.

В обновлении мы лишь добавили элемент `<p>` и неупорядоченный список `<ul>` с несколькими элементами `<li>` (с использованием списка в Python), последний из которых содержит ссылку в виде элемента `<a>`.

Обратите внимание, что все эти компоненты реализованы в Python в виде классов, в связи с чем их имена начинаются с заглавной буквы, как предписано соглашением об именовании объектов: `html.P`, `html.Ul`, `html.Li`, `html.A` и т. д.

Поэкспериментируйте самостоятельно с другими элементами и атрибутами HTML.

Проектирование макета и управление темами

Мы обсудили базовую структуру приложения Dash и рассмотрели ее основные элементы: импорт пакетов, создание экземпляра и макета приложения, функции обратного вызова (о них мы будем подробно говорить в следующей главе), а также запуск приложения. Мы также создали простейшее приложение и добавили в него несколько элементов HTML. Теперь мы готовы вывести наше приложение на новый уровень с точки зрения оформления макета. Давайте продолжим работать с атрибутом `app.layout`, но для более мощного и гибкого управления им привлечем пакет **Dash Bootstrap Components**.

Bootstrap представляет собой набор инструментов, облегчающих настройку макетов веб-страниц. Ниже мы перечислим основные преимущества и удобства, приобретаемые при использовании этого компонента:

- **темы**: как вы совсем скоро увидите, изменить тему приложения можно с помощью всего одного дополнительного аргумента при его создании. В пакете **Dash Bootstrap Components** содержится целый набор тем, которые вы можете менять и выбирать;
- **координатная сетка**: Bootstrap позволяет удобно размещать элементы приложения в соответствии с сеточным макетом с колонками и строками и не заботиться о пикселях и процентах. При этом тонкая настройка макета вам тоже останется доступна, и вы сможете воспользоваться ей при необходимости;
- **чувствительность и динамичность**: при наличии огромного множества возможных размеров экрана очень трудно бывает настроить макет приложения так, чтобы он нормально выглядел на всех устройствах. Bootstrap решает эту проблему за нас, при этом вы также можете сами управлять тем, как будут меняться размеры элементов при изменении размера экрана;
- **встроенные компоненты**: Bootstrap предлагает большое число удобных компонентов интерфейса, которыми можно при желании восполь-

зоваться. Кнопки, выпадающие меню, вкладки – это лишь малая часть того, что предлагает пакет;

- **кодировка цветов**: также в вашем распоряжении будет набор кодированных цветов, которые можно использовать при необходимости вывести ошибку, предупреждение или просто информационное сообщение для пользователя.

Давайте рассмотрим эти пункты по отдельности.

Темы

Посмотрим, как просто можно изменить тему приложения. Добавьте в наш файл `app.py` строку с импортом пакета и дополнительный аргумент в функцию создания экземпляра приложения, как показано ниже:

```
import dash_bootstrap_components as dbc
…
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.BOOOTSTRAP])
…
```

Снова запустив приложение, вы увидите, что его оформление изменилось. На рис. 1.6 представлены другие темы с указанием в нижней части страниц их названий.

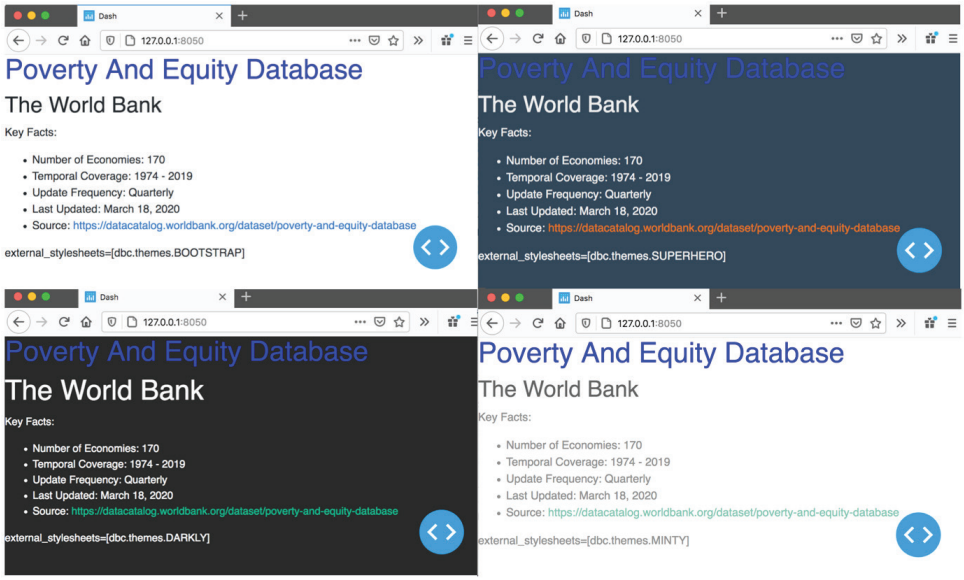

Рис. 1.6. Установка темы приложения

Как видите, достаточно одного аргумента, чтобы полностью изменить внешний вид и восприятие приложения. Также заметьте, что в элементе `<h1>` мы вручную переопределили цвет и размер шрифта при помощи аргумента `style`. Для цвета текста мы задали значение `"blue"`, а для размера – `"40px"`. Обычно

так делать не следует. Обратите внимание, как плохо читается синий текст на темных темах. Так что будьте с этим очень осторожны.

Координатная сетка и чувствительность к изменениям

Еще одно преимущество, которое дает Bootstrap, состоит в возможности использовать в приложении координатную сетку. Мы уже добавляли в элемент `html.Div` список дочерних визуальных элементов с помощью параметра `children`. В этом случае каждый элемент занимает всю доступную ширину окна, а по высоте использует минимальное место, позволяющее отобразить свое содержимое. Порядок элементов в переданном списке строго определяет их расположение на экране.

Размещение элементов в колонках

Установка всех параметров элементов посредством атрибута `style` – занятие весьма трудоемкое, а результат может оказаться не таким, как мы себе представляли. Вам необходимо учитывать слишком много факторов, и в какой-то момент все может пойти не так. С Bootstrap вам достаточно определить колонку, которая будет существовать как отдельный независимый экран, отображая размещенные в ней элементы друг под другом. При этом каждый элемент будет занимать всю доступную ширину этого «экранчика». Что касается ширины колонок-контейнеров, ее можно задать довольно гибко. Координатная сетка условно делит всю ширину экрана на 12 равных долей, и при определении ширины колонок вы можете задавать значение от 1 до 12 включительно. На рис. 1.7 показано, как могут быть определены колонки и как они могут отображаться на экранах разных размеров.

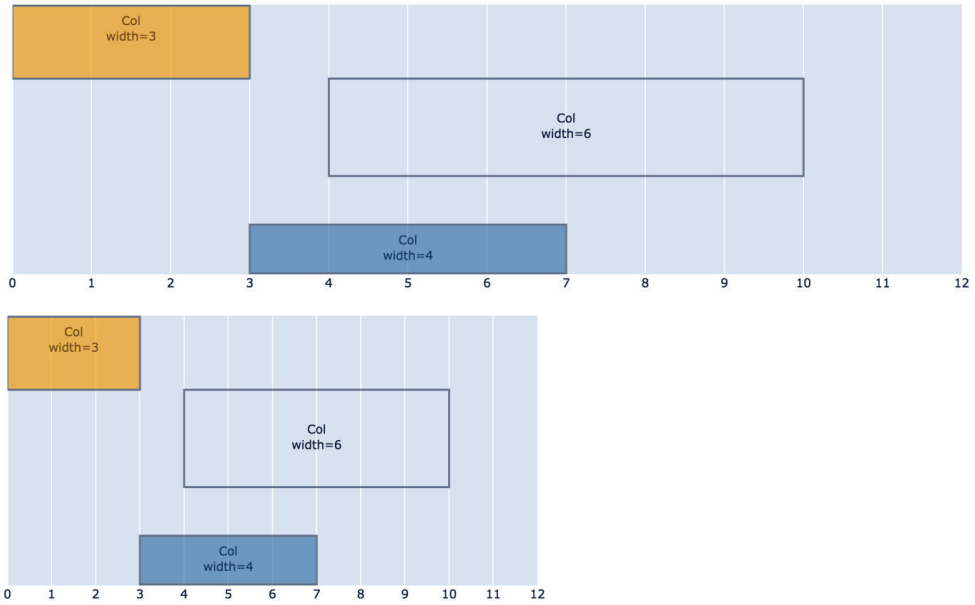

Рис. 1.7. Один и тот же колоночный макет на двух разных экранах

Как видите, размещение элементов на экране оказалось идентичным, а изменение их размеров произошло автоматически с сохранением исходных пропорций.

Но такая схема размещения элементов может не всегда вам подходить. При сильном сужении экрана может быть уместнее расширять колонки, чтобы их содержимое оставалось легкочитаемым. Это можно сделать, указав ширину колонок для пяти *вариантов ширины экрана*: `xs` (очень маленькая), `sm` (маленькая), `md` (средняя), `lg` (большая) и `xl` (очень большая). Именно так именуются и параметры, которые вы можете задать для каждой колонки с элементами.

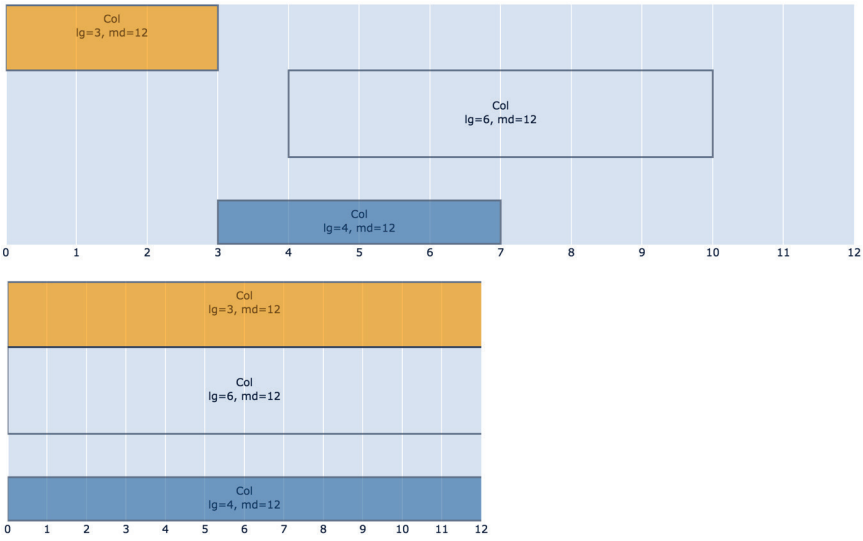

Рис. 1.8. Контроль ширины колонок в зависимости от размера экрана

На рис. 1.8 наглядно видно, как можно это реализовать путем указания двух аргументов для каждой колонки. Установить эти аргументы проще простого, и это показано ниже:

```
import dash_boostrap_components as dbc
dbc.Col(children=[child1, child2, ...], lg=6, md=12)
```

Инструкция `lg=6, md=12` означает наше желание видеть эту колонку с шириной 6 на большом экране (`lg`), что эквивалентно половине ширины экрана. На экранах со средней шириной (`md`) мы бы хотели, чтобы колонка занимала всю доступную ширину окна, для чего и установили соответствующему параметру значение 12.

Возможно, вас интересует, как нам удалось подвесить элементы на произвольном удалении от боковых границ экрана. Дело в том, что параметры размеров можно задавать при помощи словарей, одним из ключей в которых может быть **`offset`**, позволяющий указать смещение колонки от левой границы экрана, как показано ниже:

```
dbc.Col(children=[child1, child2, ...], lg={'size': 6, 'offset': 4}, md=12)
```

Как видите, параметр `lg` компонента `Col` принимает на вход словарь, в котором мы явно указали, что колонка должна размещаться с отступом в четыре позиции от левой границы.

Наконец, если вы хотите разместить колонки бок о бок, вам необходимо поместить их в одну строку (`Row`), как показано на рис. 1.9.

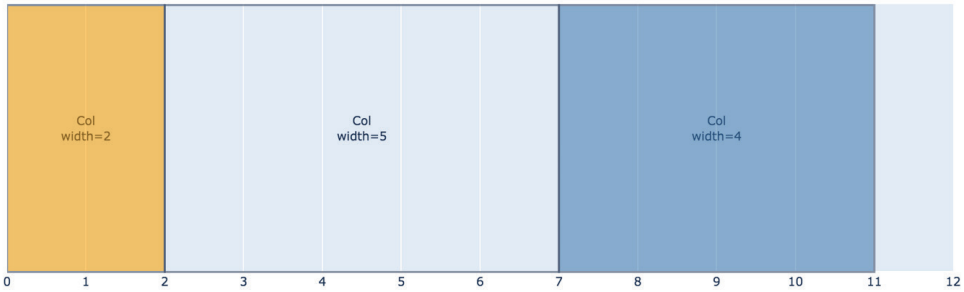

Рис. 1.9. Колонки, размещенные встык по горизонтали

Чтобы собрать такой макет, достаточно поместить все наши колонки в список и подать его на вход элементу `Row` в качестве параметра `children`, как показано ниже:

```
dbc.Row([
    dbc.Col('Column 1', width=2),
    dbc.Col('Column 2', width=5),
    dbc.Col('Column 3', width=4),
])
```

Встроенные компоненты

Конечно, мы не сможем показать в деле все встроенные компоненты Bootstrap, но некоторые из них обязательно продемонстрируем, тем более что их очень просто создавать и добавлять в приложение. Документацию по всем компонентам библиотеки можно найти по адресу **https://dash-bootstrap-components.opensource.faculty.ai**. Скоро мы покажем на примере пару встроенных компонентов.

Кодировка цветов

В своем приложении вы можете использовать любую палитру цветов на ваше усмотрение, но Bootstrap предлагает набор именованных цветов, основанных на информации, которую вы пытаетесь донести до пользователя. В некоторых компонентах вы можете установить один из таких цветов при помощи параметра `color`, и пользователю будет понятно, что вы хотите сказать. К примеру, если установить для определенного компонента параметр `color="danger"`, он станет красным, а если `color="warning"`, то желтым. Список предустановленных цветов содержит следующие значения: `primary`, `secondary`, `success`, `warning`, `danger`, `info`, `light` и `dark`.

Добавление компонентов Dash Bootstrap в приложение

Давайте добавим в наше приложение связанные компоненты `Tabs` и `Tab`. Наверное, вы уже догадались по названиям, что компонент `Tabs` является не чем иным, как контейнером для `Tab`. Все, что нам нужно, – это добавить какую-то информацию на страницы и организовать их в виде вкладок, как показано на рис. 1.10.

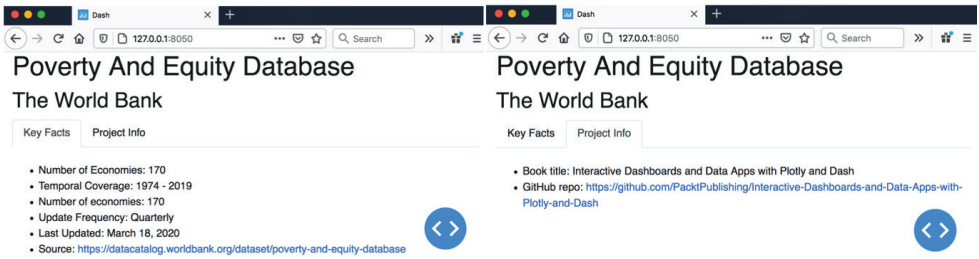

Рис. 1.10. Добавление вкладок в приложение

> **Совет**
>
> Одним из важнейших навыков при изучении фреймворка Dash является умение выполнять *рефакторинг* кода. Хотя наше приложение еще довольно простое, очень важно обладать всеми необходимыми знаниями, для того чтобы постоянно дорабатывать его без потери уже реализованного функционала. И чем больше компонентов будет в вашем приложении, тем более важную роль будет играть ваше умение держать все в голове. Я всегда советую дорабатывать приложение вручную, а не копировать код – так вы быстрее овладеете навыками рефакторинга.

Чтобы создать вкладки и наполнить их контентом, показанным на рис. 1.10, необходимо внести в наше приложение следующие изменения:

```
html.H2('The World Bank'),
dbc.Tabs([
    dbc.Tab([
        html.Ul([
            # тот же код с добавлением неупорядоченного списка
        ]),

    ], label='Key Facts'),
    dbc.Tab([
        html.Ul([
            html.Br(),
            html.Li('Book title: Interactive Dashboards and Data Apps with
Plotly and Dash'),
```

```
            html.Li(['GitHub repo: ', html.A('https://github.com/
PacktPublishing/Interactive-Dashboards-and-Data-Apps-with-Plotly-and-Dash',
href='https://github.com/PacktPublishing/Interactive-Dashboards-and-Data-
Apps-with-Plotly-and-Dash')])
    ])
], label='Project Info')
```

Полный код приложения должен выглядеть так:

```
import dash
import dash_html_components as html
import dash_bootstrap_components as dbc

app = dash.Dash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP])

app.layout = html.Div([
    html.H1('Poverty And Equity Database',
            style={'color': 'blue',
                   'fontSize': '40px'}),
    html.H2('The World Bank'),
    dbc.Tabs([
       dbc.Tab([
           html.Ul([
               html.Br(),
               html.Li('Number of Economies: 170'),
               html.Li('Temporal Coverage: 1974 - 2019'),
               html.Li('Update Frequency: Quarterly'),
               html.Li('Last Updated: March 18, 2020'),
               html.Li([
                   'Source: ',
                   html.A('https://datacatalog.worldbank.org/dataset/
poverty-and-equity-database',
                          href='https://datacatalog.worldbank.org/dataset/
poverty-and-equity-database')
               ])
           ])

       ], label='Key Facts'),
        dbc.Tab([
           html.Ul([
               html.Br(),
               html.Li('Book title: Interactive Dashboards and Data Apps
with Plotly and Dash'),
```

```
                html.Li(['GitHub repo: ',
                         html.A('https://github.com/PacktPublishing/
Interactive-Dashboards-and-Data-Apps-with-Plotly-and-Dash',
                                href='https://github.com/PacktPublishing/
Interactive-Dashboards-and-Data-Apps-with-Plotly-and-Dash')
                         ])
            ])
        ], label='Project Info')
    ]),
])


if __name__ == '__main__':
    app.run_server(debug=True)
```

Как видите, мы добавили в приложение один элемент `Tabs`, в котором разместили два элемента `Tab`. В первом из них мы оставили написанный ранее код с неупорядоченным списком, а во втором также поместили список с другим содержимым. Ну, ладно, это содержимое вы можете и скопировать! Также обратите внимание, как задаются названия вкладок при помощи параметра `label`.

Теперь вы можете запустить обновленное приложение и убедиться, что вкладки отображаются так, как вы и предполагали[2].

Что ж, пришло время добавить нашему приложению интерактивности!

Заключение

Основным в этой главе было то, что вы научились создавать приложения Dash и даже успели понять, насколько это на самом деле просто. Также мы обсудили содержимое базовых пакетов фреймворка Dash, служащих для добавления визуальных элементов в приложение. Вы приобрели достаточно знаний, чтобы создавать даже сложные макеты с размещением на них любого числа элементов. При этом мы пока не делали ничего сложного и далее в этой книге продолжим обсуждать эти и другие компоненты, чтобы вы могли набить руку и научиться лучше обращаться с ними.

В следующей главе мы попробуем оживить добавленные в приложение элементы, придав им интерактивности. Мы изменим приложение таким образом, чтобы пользователь мог сам делать выбор и анализировать данные в том разрезе, в котором ему необходимо.

[2] Приложение проверено на версиях фреймворка Dash 1.19.0 и актуальной на момент перевода книги – 2.4.1. – *Прим. перев.*