**Разработка интерфейсов визуализации данных на Python с помощью Dash**

Прежде чем приступить к выполнению работы, давайте разберем, что же такое Dash? [Dash](https://dash.plotly.com/introduction) — это платформа с открытым исходным кодом для создания интерфейсов визуализации данных, помогающая специалистам по обработке данных создавать аналитические веб-приложения, не требуя передовых знаний в области веб-разработки.

Я всегда работаю в PyCharm, поэтому, прежде чем приступить к работе, я создаю проект, в котором создается виртуальная среда. Далее, я добавляю в проект файл, с которым будет происходить работа, а именно: avocado.csv, содержащий набор данных о продажах и ценах на авокадо в США за период с 2015 по 2018 годы. Скачать данный файл можно [здесь](https://github.com/realpython/materials/blob/master/python-dash/additional_files/avocado.csv). После этого, необходимо установить следующие библиотеки: dash и pandas. Теперь мы можем приступать к работе!

**Создание первого приложения**

Давайте создадим пустой файл с именем **app1.py**в корневом каталоге нашего проекта и создадим наше первое приложение:

import dash  
import dash\_core\_components as dcc  
import dash\_html\_components as html  
import pandas as pd  
  
data = pd.read\_csv("avocado.csv")  
data = data.query("type == 'conventional' and region == 'Albany'")  
data["Date"] = pd.to\_datetime(data["Date"], format="%Y-%m-%d")  
data.sort\_values("Date", inplace=True)  
  
app = dash.Dash(\_\_name\_\_)  
  
app.layout = html.Div(  
 children=[  
 html.H1(  
 children="Avocado Analytics",  
 style={"fontSize": "48px", "color": "red"},  
 ),  
 html.P(  
 children="Analyze the behavior of avocado prices"  
 " and the number of avocados sold in the US"  
 " between 2015 and 2018",  
 ),  
 dcc.Graph(  
 figure={  
 "data": [  
 {  
 "x": data["Date"],  
 "y": data["AveragePrice"],  
 "type": "lines",  
 },  
 ],  
 "layout": {"title": "Average Price of Avocados"},  
 },  
 ),  
 dcc.Graph(  
 figure={  
 "data": [  
 {  
 "x": data["Date"],  
 "y": data["Total Volume"],  
 "type": "lines",  
 },  
 ],  
 "layout": {"title": "Avocados Sold"},  
 },  
 ),  
 ]  
)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run\_server(debug=True)

В строках с 1 по 4 мы импортируем необходимые библиотеки: dash, dash\_core\_components, dash\_html\_components и pandas. Каждая библиотека предоставляет стандартный блок для вашего приложения:

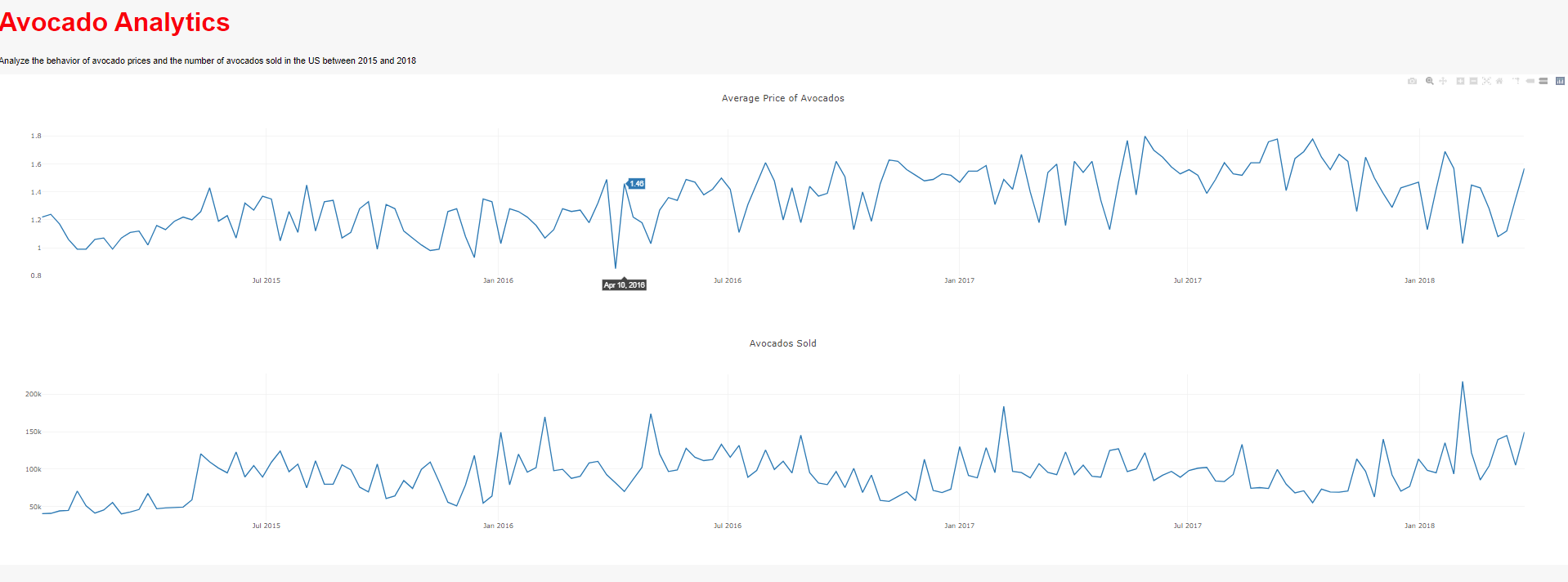
* dash поможет нам инициализировать ваше приложение.
* dash\_core\_components позволяет создавать интерактивные компоненты, такие как графики, раскрывающиеся списки или диапазоны дат.
* dash\_html\_components позволяет получить доступ к тегам HTML.
* pandas помогает читать и систематизировать данные.

В строках с 6 по 9 мы считываем данные и предварительно обрабатываем их для использования на панели управления. Мы фильтруем некоторые данные, потому что текущая версия нашей информационной панели не является интерактивной и в противном случае значения на графике не имели бы смысла. В строке 11, мы создаем экземпляр класса Dash. Если вы раньше использовали Flask, то инициализация класса Dash может показаться вам знакомой. В Flask вы обычно инициализируете приложение WSGI с помощью Flask(\_\_name\_\_). Точно так же для приложения Dash вы используете Dash(\_\_name\_\_).

Далее, мы определяем свойство макета нашего приложения, которое определяет внешний вид нашего приложения. В нашем случае мы будем использовать заголовок с описанием под ним и двумя графиками В строках с 13 по 23 вы можете увидеть на практике компоненты Dash HTML. Мы начинаем с определения родительского компонента, html.Div. Затем мы добавляем еще два элемента, заголовок (html.H1) и абзац (html.P) в качестве его дочерних элементов. Эти компоненты эквивалентны div, HTML-теги h1 и p. Вы можете использовать аргументы компонентов для изменения атрибутов или содержимого тегов. Например, чтобы указать, что находится внутри тега div, мы используем аргумент children в html.Div. В компонентах также есть другие аргументы, такие как style, className или id, которые относятся к атрибутам тегов HTML. При этом, если вы хотите изменить размер и цвет элемента H1, то достаточно прописать изменения через style, как это сделал я в 17 строке.

В строках 24–50 фрагмента кода макета вы можете увидеть компонент графика из Dash Core Components на практике. В app.layout есть два компонента dcc.Graph. Первый отображает средние цены на авокадо за период исследования, а второй — количество авокадо, проданных в США за тот же период. Под капотом Dash использует Plotly.js для создания графиков. Компоненты dcc.Graph ожидают объект фигуры или словарь Python, содержащий данные графика и макет. В этом случае вы предоставляете последнее.

И последние две строчки кода позволяют запускать приложение Dash локально, используя встроенный сервер Flask. Далее, мы запускаем наше приложение, после чего переходим по высветившейся ссылке и смотрим результат:



**Создание второго приложения с добавлением внешних ресурсов**

В данной главе вы узнаете, как настроить внешний вид панели инструментов, а именно:

* Добавление значка и заголовка на страницу.
* Изменение семейства шрифтов вашей панели инструментов.
* Использование внешних файлов CSS для стилизации компонентов Dash.

Сначала, в папке с проектом, мы создаем еще одну папку с именем assets, в которую добавляем скаченный значок с именем favicon.ico (скачать его можно [здесь](https://raw.githubusercontent.com/dylanjcastillo/materials/python-dash/python-dash/additional_files/favicon.ico)) и создаем файл style.css, содержащий следующие команды:

body {  
 font-family: "Lato", sans-serif;  
 margin: 0;  
 background-color: #F7F7F7;  
}  
  
.header {  
 background-color: #222222;  
 height: 256px;  
 display: flex;  
 flex-direction: column;  
 justify-content: center;  
}  
  
.header-emoji {  
 font-size: 48px;  
 margin: 0 auto;  
 text-align: center;  
}  
  
.header-title {  
 color: #FFFFFF;  
 font-size: 48px;  
 font-weight: bold;  
 text-align: center;  
 margin: 0 auto;  
}  
  
.header-description {  
 color: #CFCFCF;  
 margin: 4px auto;  
 text-align: center;  
 max-width: 384px;  
}  
  
.wrapper {  
 margin-right: auto;  
 margin-left: auto;  
 max-width: 1024px;  
 padding-right: 10px;  
 padding-left: 10px;  
 margin-top: 32px;  
}  
  
.card {  
 margin-bottom: 24px;  
 box-shadow: 0 4px 6px 0 rgba(0, 0, 0, 0.18);  
}

Файлы в папке **assets** содержат стили, которые мы применим к компонентам в макете приложения. После запуска сервера, Dash автоматически обслуживает файлы, расположенные в **этой папке**. И для установки значка по умолчанию нам не нужно предпринимать никаких дополнительных действий. Для использования стилей, определенных в style.css, нам необходимо использовать аргумент className в компонентах Dash. Поэтому, давайте создадим новый файл app2.py, в котором пропишем следующий код:

import dash  
import dash\_core\_components as dcc  
import dash\_html\_components as html  
import pandas as pd  
  
data = pd.read\_csv("avocado.csv")  
data = data.query("type == 'conventional' and region == 'Albany'")  
data["Date"] = pd.to\_datetime(data["Date"], format="%Y-%m-%d")  
data.sort\_values("Date", inplace=True)  
  
external\_stylesheets = [  
 {  
 "href": "https://fonts.googleapis.com/css2?"  
 "family=Lato:wght@400;700&display=swap",  
 "rel": "stylesheet",  
 },  
]  
app = dash.Dash(\_\_name\_\_, external\_stylesheets=external\_stylesheets)  
app.title = "Avocado Analytics: Understand Your Avocados!"  
  
app.layout = html.Div(  
 children=[  
 html.Div(  
 children=[  
 html.P(children="🥑", className="header-emoji"),  
 html.H1(  
 children="Avocado Analytics", className="header-title"  
 ),  
 html.P(  
 children="Analyze the behavior of avocado prices"  
 " and the number of avocados sold in the US"  
 " between 2015 and 2018",  
 className="header-description",  
 ),  
 ],  
 className="header",  
 ),  
 html.Div(  
 children=[  
 html.Div(  
 children=dcc.Graph(  
 id="price-chart",  
 config={"displayModeBar": False},  
 figure={  
 "data": [  
 {  
 "x": data["Date"],  
 "y": data["AveragePrice"],  
 "type": "lines",  
 "hovertemplate": "$%{y:.2f}"  
 "<extra></extra>",  
 },  
 ],  
 "layout": {  
 "title": {  
 "text": "Average Price of Avocados",  
 "x": 0.05,  
 "xanchor": "left",  
 },  
 "xaxis": {"fixedrange": True},  
 "yaxis": {  
 "tickprefix": "$",  
 "fixedrange": True,  
 },  
 "colorway": ["#17B897"],  
 },  
 },  
 ),  
 className="card",  
 ),  
 html.Div(  
 children=dcc.Graph(  
 id="volume-chart",  
 config={"displayModeBar": False},  
 figure={  
 "data": [  
 {  
 "x": data["Date"],  
 "y": data["Total Volume"],  
 "type": "lines",  
 },  
 ],  
 "layout": {  
 "title": {  
 "text": "Avocados Sold",  
 "x": 0.05,  
 "xanchor": "left",  
 },  
 "xaxis": {"fixedrange": True},  
 "yaxis": {"fixedrange": True},  
 "colorway": ["#E12D39"],  
 },  
 },  
 ),  
 className="card",  
 ),  
 ],  
 className="wrapper",  
 ),  
 ]  
)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run\_server(debug=True)

Код точно такой же, как и в app1.py, но с некоторыми изменениями. В строках с 21 по 37 вы можете увидеть, что в исходную версию панели мониторинга были внесены два изменения:

1. Появился новый элемент абзаца с эмодзи авокадо, который будет служить логотипом.
2. В каждом компоненте есть аргумент className. Эти имена классов должны соответствовать селектору классов в style.css, который будет определять внешний вид каждого компонента.

Например, класс header-description, назначенный компоненту абзаца, начинающемуся с «Analyze the behavior of avocado prices», имеет соответствующий селектор в style.css:

.header-description {

color: #CFCFCF;

margin: 4px auto;

text-align: center;

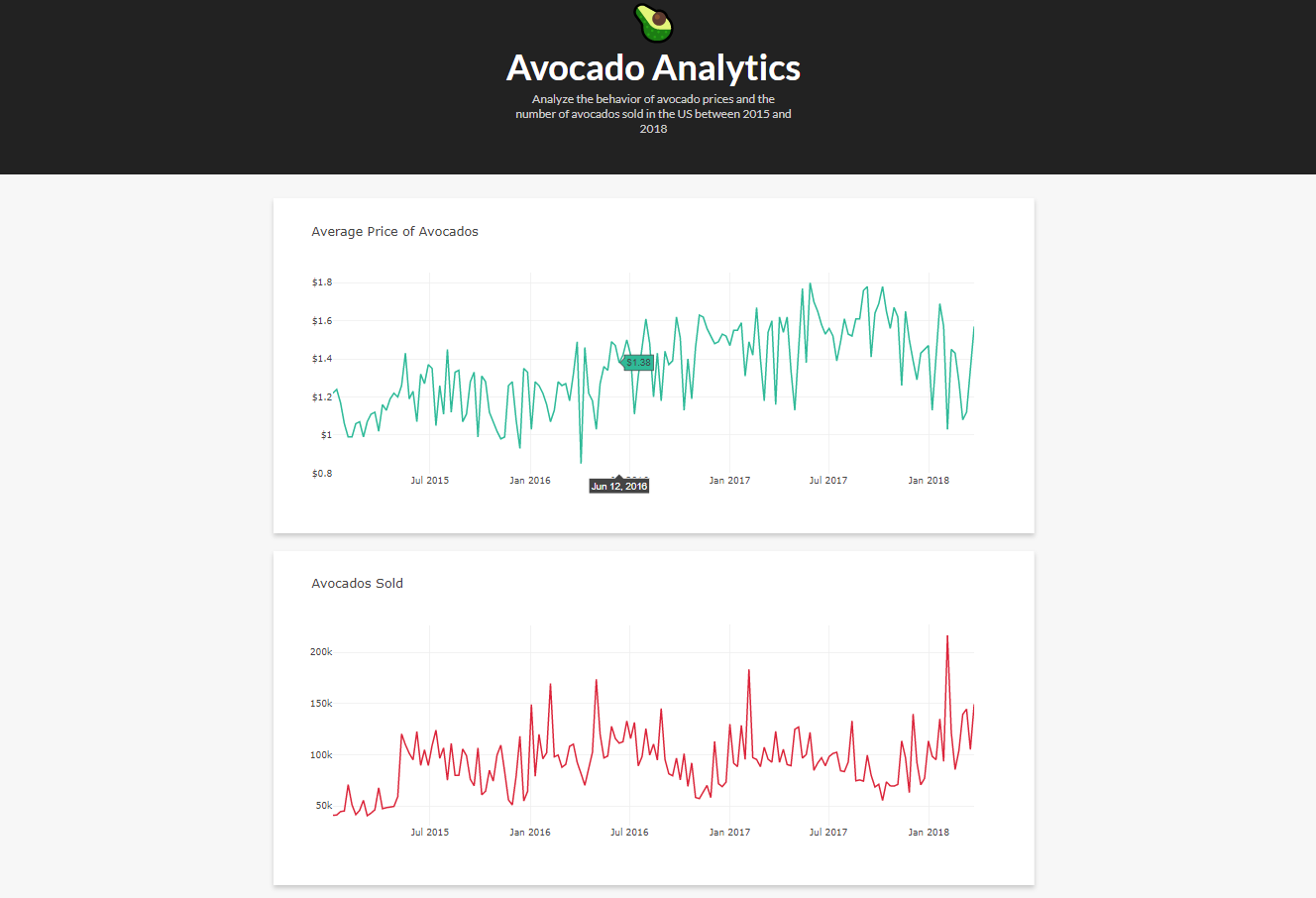
max-width: 384px;

}

Строки с 29 по 34 файла style.css определяют формат селектора класса описания заголовка. Они изменят цвет, поля, выравнивание и максимальную ширину любого компонента с className = "header-description". Все компоненты имеют соответствующие селекторы классов в файле CSS. Другое существенное изменение — графики. При построении графика цены, мы выполнили следующие изменения:

* Строка 43: мы удаляем плавающую полосу, которая отображается на графике по умолчанию.
* Строки 50 и 51: мы устанавливаем шаблон наведения таким образом, чтобы при наведении курсора на точку данных отображалась цена в долларах. Вместо 2,5он будет отображаться как 2,5 доллара.
* Строки с 54 по 66: мы настраиваем ось, цвет рисунка и формат заголовка в разделе макета графика.
* Строка 69: мы оборачиваем график в html.Div с классом «card». Это придаст графику белый фон и добавит небольшую тень под ним.

Аналогичные изменения внесены в графики продаж и объемов. После всех этих изменений наша панель управления должна выглядеть так:



## Добавление интерактивности в приложения Dash с помощью обратных вызовов

В этом разделе мы будем добавлять интерактивные элементы на панель управления. Интерактивность Dash основана на парадигме реактивного программирования. Это означает, что вы можете связывать компоненты с элементами вашего приложения, которые хотите обновить. Если пользователь взаимодействует с компонентом ввода, например, с раскрывающимся списком или ползунком диапазона, то вывод, например, график, будет автоматически реагировать на изменения во вводе. Теперь давайте сделаем вашу панель управления интерактивной. Эта новая версия вашей панели инструментов позволит пользователю взаимодействовать со следующими фильтрами:

* Область
* Тип авокадо
* Диапазон дат

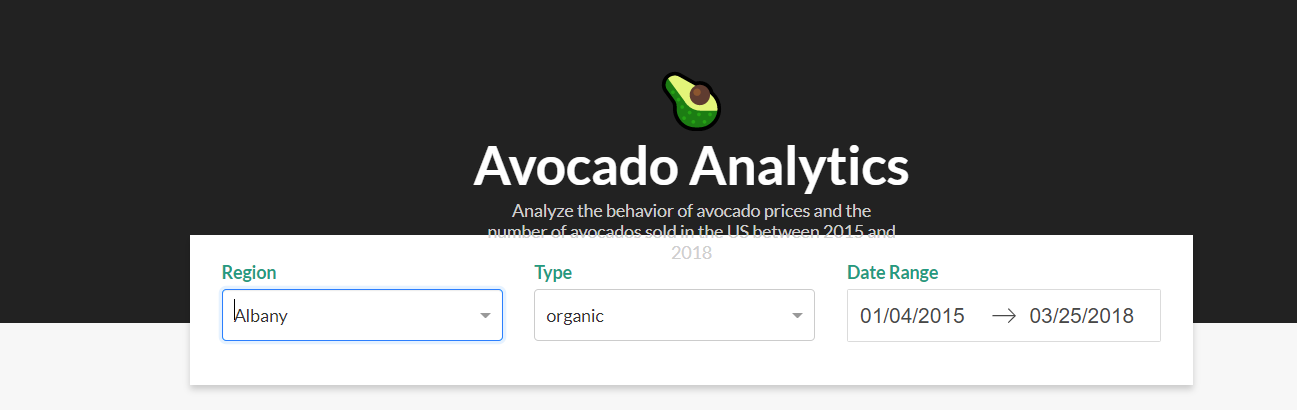
Поэтому, давайте создадим новый файл в папке assets с имеем style2.css и пропишем следующие команды:

body {  
 font-family: "Lato", sans-serif;  
 margin: 0;  
 background-color: #F7F7F7;  
}  
  
.header {  
 background-color: #222222;  
 height: 288px;  
 padding: 16px 0 0 0;  
}  
  
.header-emoji {  
 font-size: 48px;  
 margin: 0 auto;  
 text-align: center;  
}  
  
.header-title {  
 color: #FFFFFF;  
 font-size: 48px;  
 font-weight: bold;  
 text-align: center;  
 margin: 0 auto;  
}  
  
.header-description {  
 color: #CFCFCF;  
 margin: 4px auto;  
 text-align: center;  
 max-width: 384px;  
}  
  
.wrapper {  
 margin-right: auto;  
 margin-left: auto;  
 max-width: 1024px;  
 padding-right: 10px;  
 padding-left: 10px;  
 margin-top: 32px;  
}  
  
.card {  
 margin-bottom: 24px;  
 box-shadow: 0 4px 6px 0 rgba(0, 0, 0, 0.18);  
}  
  
.menu {  
 height: 112px;  
 width: 912px;  
 display: flex;  
 justify-content: space-evenly;  
 padding-top: 24px;  
 margin: -80px auto 0 auto;  
 background-color: #FFFFFF;  
 box-shadow: 0 4px 6px 0 rgba(0, 0, 0, 0.18);  
}  
  
.Select-control {  
 width: 256px;  
 height: 48px;  
}  
  
.Select--single > .Select-control .Select-value, .Select-placeholder {  
 line-height: 48px;  
}  
  
.Select--multi .Select-value-label {  
 line-height: 32px;  
}  
  
.menu-title {  
 margin-bottom: 6px;  
 font-weight: bold;  
 color: #079A82;  
}

После этого, мы создаем еще один файл, но уже просто в папке с проектом, с именем app3.py, в котором прописываем следующий код:

import dash  
import dash\_core\_components as dcc  
import dash\_html\_components as html  
import pandas as pd  
import numpy as np  
from dash.dependencies import Output, Input  
  
data = pd.read\_csv("avocado.csv")  
data["Date"] = pd.to\_datetime(data["Date"], format="%Y-%m-%d")  
data.sort\_values("Date", inplace=True)  
  
external\_stylesheets = [  
 {  
 "href": "https://fonts.googleapis.com/css2?"  
 "family=Lato:wght@400;700&display=swap",  
 "rel": "stylesheet",  
 },  
]  
app = dash.Dash(\_\_name\_\_, external\_stylesheets=external\_stylesheets)  
app.title = "Avocado Analytics: Understand Your Avocados!"  
  
app.layout = html.Div(  
 children=[  
 html.Div(  
 children=[  
 html.P(children="🥑", className="header-emoji"),  
 html.H1(  
 children="Avocado Analytics", className="header-title"  
 ),  
 html.P(  
 children="Analyze the behavior of avocado prices"  
 " and the number of avocados sold in the US"  
 " between 2015 and 2018",  
 className="header-description",  
 ),  
 ],  
 className="header",  
 ),  
 html.Div(  
 children=[  
 html.Div(  
 children=[  
 html.Div(children="Region", className="menu-title"),  
 dcc.Dropdown(  
 id="region-filter",  
 options=[  
 {"label": region, "value": region}  
 for region in np.sort(data.region.unique())  
 ],  
 value="Albany",  
 clearable=False,  
 className="dropdown",  
 ),  
 ]  
 ),  
 html.Div(  
 children=[  
 html.Div(children="Type", className="menu-title"),  
 dcc.Dropdown(  
 id="type-filter",  
 options=[  
 {"label": avocado\_type, "value": avocado\_type}  
 for avocado\_type in data.type.unique()  
 ],  
 value="organic",  
 clearable=False,  
 searchable=False,  
 className="dropdown",  
 ),  
 ],  
 ),  
 html.Div(  
 children=[  
 html.Div(  
 children="Date Range",  
 className="menu-title"  
 ),  
 dcc.DatePickerRange(  
 id="date-range",  
 min\_date\_allowed=data.Date.min().date(),  
 max\_date\_allowed=data.Date.max().date(),  
 start\_date=data.Date.min().date(),  
 end\_date=data.Date.max().date(),  
 ),  
 ]  
 ),  
 ],  
 className="menu",  
 ),  
 html.Div(  
 children=[  
 html.Div(  
 children=dcc.Graph(  
 id="price-chart", config={"displayModeBar": False},  
 ),  
 className="card",  
 ),  
 html.Div(  
 children=dcc.Graph(  
 id="volume-chart", config={"displayModeBar": False},  
 ),  
 className="card",  
 ),  
 ],  
 className="wrapper",  
 ),  
 ]  
)  
  
  
@app.callback(  
 [Output("price-chart", "figure"), Output("volume-chart", "figure")],  
 [  
 Input("region-filter", "value"),  
 Input("type-filter", "value"),  
 Input("date-range", "start\_date"),  
 Input("date-range", "end\_date"),  
 ],  
)  
def update\_charts(region, avocado\_type, start\_date, end\_date):  
 mask = (  
 (data.region == region)  
 & (data.type == avocado\_type)  
 & (data.Date >= start\_date)  
 & (data.Date <= end\_date)  
 )  
 filtered\_data = data.loc[mask, :]  
 price\_chart\_figure = {  
 "data": [  
 {  
 "x": filtered\_data["Date"],  
 "y": filtered\_data["AveragePrice"],  
 "type": "lines",  
 "hovertemplate": "$%{y:.2f}<extra></extra>",  
 },  
 ],  
 "layout": {  
 "title": {  
 "text": "Average Price of Avocados",  
 "x": 0.05,  
 "xanchor": "left",  
 },  
 "xaxis": {"fixedrange": True},  
 "yaxis": {"tickprefix": "$", "fixedrange": True},  
 "colorway": ["#17B897"],  
 },  
 }  
  
 volume\_chart\_figure = {  
 "data": [  
 {  
 "x": filtered\_data["Date"],  
 "y": filtered\_data["Total Volume"],  
 "type": "lines",  
 },  
 ],  
 "layout": {  
 "title": {"text": "Avocados Sold", "x": 0.05, "xanchor": "left"},  
 "xaxis": {"fixedrange": True},  
 "yaxis": {"fixedrange": True},  
 "colorway": ["#E12D39"],  
 },  
 }  
 return price\_chart\_figure, volume\_chart\_figure  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run\_server(debug=True)

Данный код очень схож с app1 и app2, но он так же претерпел изменения. В строках с 24 по 74 мы определяем html.Div поверх наших графиков, состоящих из двух раскрывающихся списков и селектора диапазона дат. Он будет служить меню, которое пользователь будет использовать для взаимодействия с данными:



В строках с 41 по 55 мы определяем раскрывающийся список, который пользователи будут использовать для фильтрации данных по региону. Помимо заголовка, в нем есть компонент dcc.Dropdown. Вот что означает каждый из параметров:

* id — идентификатор элемента.
* options — это параметры, отображаемые при выборе раскрывающегося списка. Он ожидает словарь с метками и значениями.
* value — значение по умолчанию при загрузке страницы.
* clearable позволяет пользователю оставить это поле пустым, если установлено значение True.
* className — это селектор классов, используемый для применения стилей.

Селекторы «Type» и «Date Range» имеют ту же структуру, что и раскрывающееся меню «Region».

В строках с 90 по 106 мы определяем компоненты dcc.Graph. Возможно, вы заметили, что по сравнению с предыдущей версией панели инструментов в компонентах отсутствует аргумент figure. Это связано с тем, что аргумент figure теперь будет генерироваться функцией обратного вызова с использованием входных данных, которые пользователь устанавливает с помощью селекторов «Region», «Type» и «Date Range».

Но пока что мы только определили, как пользователь будет взаимодействовать с нашим приложением. Теперь нам нужно заставить ваше приложение реагировать на действия пользователя. Для этого вы воспользуетесь функциями обратного вызова. Функции обратного вызова Dash — это обычные функции Python с декоратором app.callback. В Dash при изменении ввода запускается функция обратного вызова. Функция выполняет некоторые заранее определенные операции, такие как фильтрация набора данных, и возвращает результат в приложение. По сути, обратные вызовы связывают входы и выходы в вашем приложении. Сама функция обратного вызова, используемая для обновления графиков, представлена со строки 111 по 164.

В строках 111–119 мы определяем входы и выходы внутри декоратора app.callback. Сначала мы определяем выходы с помощью объектов вывода. Эти объекты принимают два аргумента:

1. Идентификатор элемента, который они изменят при выполнении функции.
2. Свойство изменяемого элемента.

Например, Output("price-chart", "figure") обновит свойство figure элемента «price-chart». Затем мы определяем входы с помощью объектов Input. Они также принимают два аргумента:

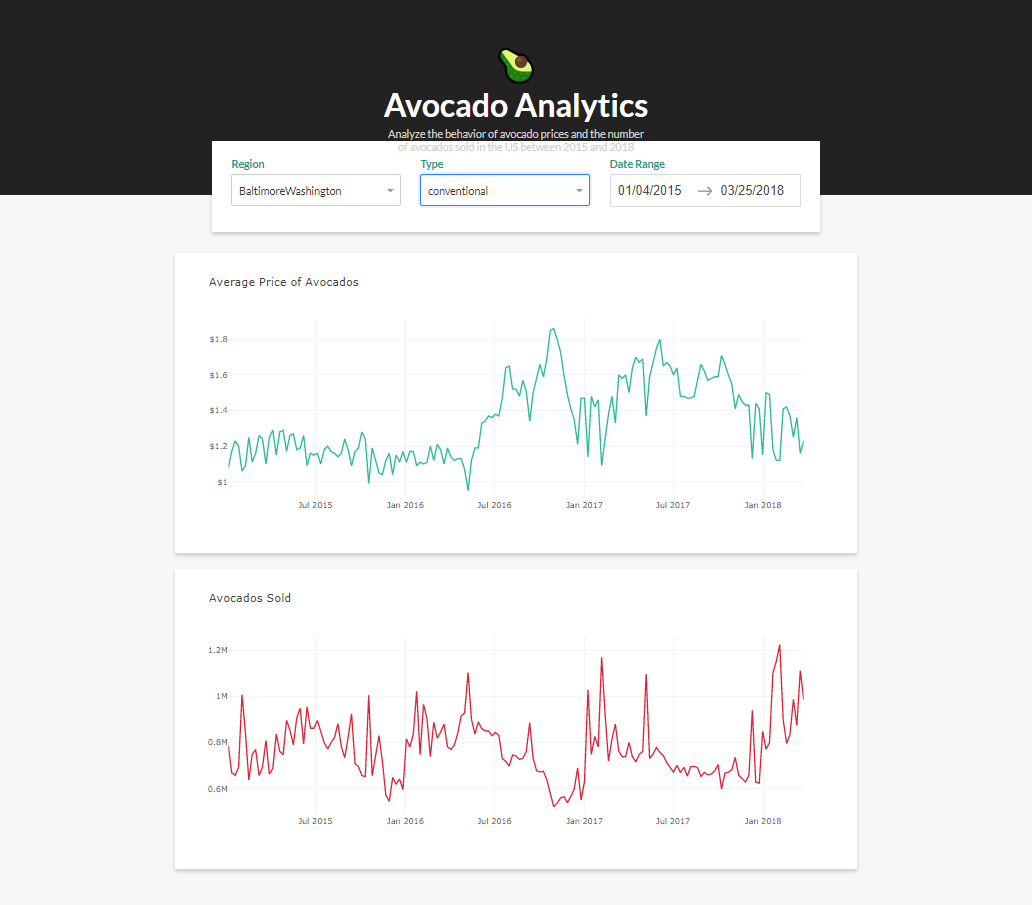
1. Идентификатор элемента, за изменениями которого они будут следить.
2. Свойство наблюдаемого элемента, которое они должны принимать, когда происходит изменение.

Итак, Input("region-filter", "value") будет следить за элементом «region-filter» на предмет изменений и примет его свойство value, если элемент изменится.

В строке 120 мы определяем функцию, которая будет применяться при изменении ввода. Здесь следует отметить, что аргументы функции будут соответствовать порядку объектов Input, переданных в обратный вызов. Нет явной связи между именами аргументов в функции и значениями, указанными во входных объектах.

Наконец, в строках 121–164 вы определяете тело функции. В этом случае функция принимает входные данные (region, type of avocado и date range), фильтрует данные и генерирует объекты-фигуры для графиков цен и объемов.

Результат:



Исходя из всего проделанного выше, мы можем использовать Dash для создания аналитических приложений, что на данный момент очень востребовано и ценно в современном мире.