

# Part II

## My First Dash App

# 1 Twitter Analysis of Likes

---

## 1.1 What You Will Learn

So, this is your first app... This is so exciting!

In this first web app dashboard, we will be analyzing the amount of likes that 16 celebrities have received on their Twitter posts since 2011. This type of twitter analysis is very common in the field of social media analytics. It is typically used to better understand your audience behavior on social media, the effectiveness of your social media posts, and how your account is performing overall. Given that this is our first app, we will start by plotting only the amount of likes per tweet; however, once you master this simple plotting process with Dash Plotly, you will be on your way to plot bigger and more complex data such as: Instagram post views, or Facebook profile visits, or LinkedIn posts' click-through-rates (the number of clicks divided by the number of times posts is shown).

By the end of Part II, you will have sufficient knowledge of Dash Plotly to create your own web app dashboard. You will learn how to incorporate data into your app, manage numerous web app components (`dropdown`, `figure`, `Div`), build basic charts such as the line chart, and most importantly, add interactive capabilities to your dashboard through the callback decorator. But first thing first – let's set up the app and the corresponding code.

## 1.2 Creation of Project and Code Execution

The code below assumes that you have a Python IDE installed on your computer. If that is not the case, please refer back up to *Part I* of this book to complete your Python IDE setup so you can execute this app successfully.

To get started, you will need to create your project and download the necessary files into that project. First, create a new folder called “my-first-app”. We will refer to this folder as your Project Folder. Now, download the tweets.csv<sup>1</sup> file into your project folder, using this link:


[https://1drv.ms/u/s!AoKPiuofdvp\\_goUHLluEljohtC5GsQ?e=eeYPmu](https://1drv.ms/u/s!AoKPiuofdvp_goUHLluEljohtC5GsQ?e=eeYPmu)

Next, download the “twitter\_app.py” file into the same project folder:

[https://github.com/DashBookProject/Dash-Plotly/blob/master/Code/Part%20I/twitter\\_app.py](https://github.com/DashBookProject/Dash-Plotly/blob/master/Code/Part%20I/twitter_app.py)

The project folder should have the tweets.csv file and the twitter\_app.py file, and it should look like this:

```
- my-first-app
  | -- tweets.csv
  | -- twitter_app.py
```

Now that you have your project folder all set up, let’s install the necessary Python libraries for this app to work properly. Open your  Command Prompt or the Terminal inside your Python IDE. CD into (go to) the project folder you just created, and install Numpy, Pandas, and Dash (the Plotly package is automatically installed with Dash). To install these libraries, just type inside the terminal or Command Prompt:

```
$ pip install numpy
$ pip install pandas
$ pip install dash
```

This process should take approximately 10 minutes. Once you have the libraries installed, open your twitter\_app.py file inside your Python IDE and run the script. You will see a message that says:

---

<sup>1</sup> Source: <https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/JBXKFD>

```
* Serving Flask app "twitter_app" (lazy loading)
* Environment: production

WARNING: This is a development server. Do not use it in a production
deployment.

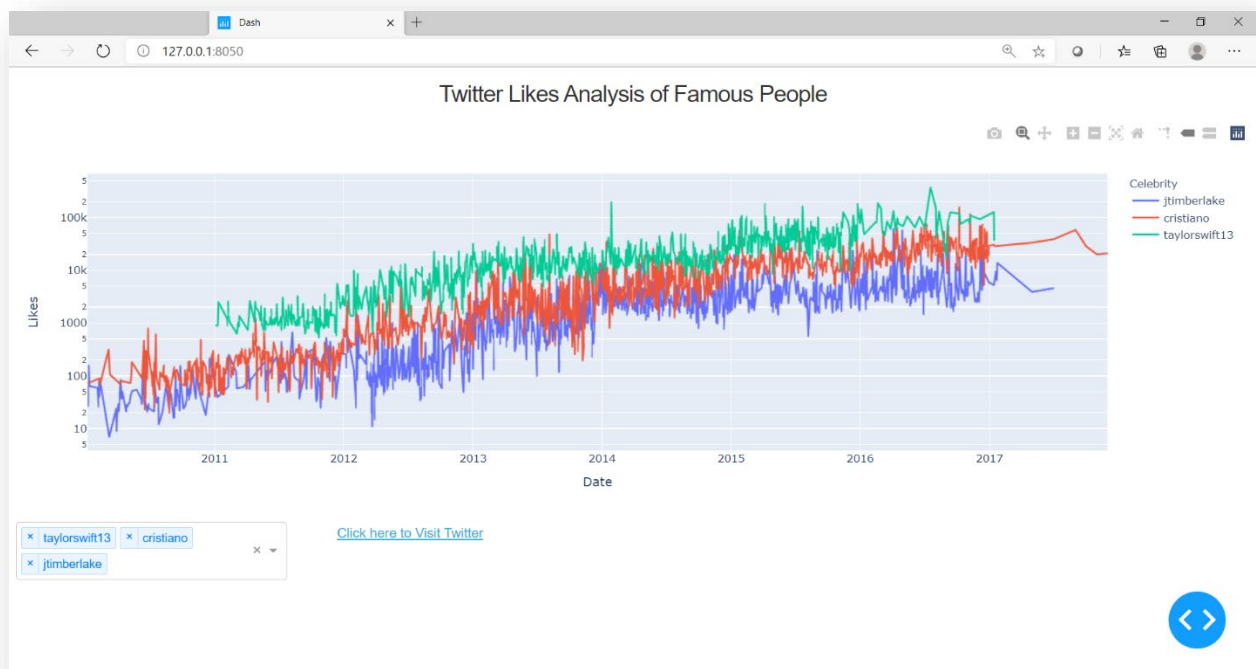
Use a production WSGI server instead.

* Debug mode: on

Dash is running on http://127.0.0.1:8050/
```

The message content and warning are completely normal – don't worry. Just click on the http link to open your app inside your default browser.

**Congratulations!** You should now see your first Dash app, which should look like this:



Have fun! Play around with your web app dashboard. Change the Dropdown values, click on the link, click on the graph legend at the top right, zoom into a certain date range by holding down the left-click button of the mouse.

## 1.3 Python Libraries

Before we dive into the data management section of the code, let's go over the first couple of rows to better understand the libraries being used.

```
import pandas as pd
import plotly.express as px

import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output

twitter_app.py file: rows 1 – 7
```

There are two main ways to create graphs in Plotly: 1. [Plotly Graph Objects](#) and 2. [Plotly Express](#). Plotly Graph Objects is a low-level interface to create graphs from the bottom-up. When using Graph Objects, you need to define the Data, the Layout, and at times, the Frames, all of which makes the graph-building process somewhat laborious compared to Plotly Express. That said, its full set of attributes allow you to customize your graphs in ways that add much richness to them.

Plotly Express, on the other hand, is a high-level interface to create graphs in single-function calls, with very few lines of code. And it has a nice amount of attributes that allow you to build graphs seamlessly and quickly. For these reasons, we will be using Plotly Express in most cases and revert to Graph Objects only when needed.

The [dash core components](#) and the [dash html components](#) are one of the most important elements in Dash. They both refer to Dash components that form an essential part of any dashboard being built. These components include the `Dropdown`, the `Checkbox`, the `RangeSlider`, the `Button`, and many more features that are used to connect the data to the graphs. We will review them thoroughly in chapter 3. Lastly, the *Input* and *Output* are the building blocks of the callback decorator, which we will review at length in chapter 5.

## 1.4 Data Management

There are many ways to accumulate and store data. Using a CSV spreadsheet or an Excel sheet is one way, but other commonly used methods are: SQL databases such as PostgreSQL, Amazon Redshift, Apache Spark, Microsoft SQL Server, and MySQL; NoSQL databases such as MongoDB and Redis also exist. Another way to gather data is to scrape the web or import data from third parties by connecting to an API. A popular

usage of an API connection to a dashboard is finance related. Many dashboards visualize financial data such as stock prices by pulling the latest prices via an API, which in turn keeps the data fresh and the dashboard consistently updated.

In this web app dashboard, we are using a CSV spreadsheet as our data source. But to use the data, we need to find a way to read it into the app. For that, we have Pandas (Python library explained thoroughly in Part I if you need a refresher). The first line of the code below takes the CSV spreadsheet and reads it into a Pandas dataframe called “df”.

```
df = pd.read_csv("tweets.csv")
df["name"] = pd.Series(df["name"]).str.lower()
df["date_time"] = pd.to_datetime(df['date_time']).dt.date
df = df.groupby(['date_time', "name"]) [ ["number_of_likes",
                                         "number_of_shares"] ].mean().astype(int)
df = df.reset_index()
twitter_app.py file: rows 11 – 16
```

Once we have the data inside a Pandas dataframe, it's a lot easier to clean, filter, sort, and manipulate for the purpose of the app. In this app we need to prepare the data for the plotting of the line chart. Therefore, we do the following in the code above: we change the strings of the celebrity *name* column to lowercase (second line of code); we convert the *date\_time* column into a date recognizable by Pandas (third line of code); and we group the data by *date\_time* and *name*, because some celebrities have multiple tweets on the same day (fourth line of code).

If you `print(df.head())`, you should get the table below to print inside the Python terminal. As you can see, the result is a beautiful Pandas dataframe with rows of data that represent the average number of likes per celebrity, per day.

	date_time	name	number_of_likes	number_of_shares
0	2010-01-06	selenagomez	278	695
1	2010-01-07	jtimberlake	62	189
2	2010-01-07	selenagomez	201	630
3	2010-01-08	jtimberlake	27	107
4	2010-01-08	selenagomez	349	935

**\*\*It is important to note that you should always read your data and prepare it for the app at the beginning of the app – before you write out the layout section. Reading data into**

the app can be a memory-expensive task; by inserting the data at the beginning part of the app, you ensure that the app loads the data into memory only once and does not repeat this process every time a user interacts with the dashboard through the callback. A dataframe (df) at the beginning part of the app is commonly referred to as a **global dataframe** or data that is in a **global state**.

In cases where you would like to test the plotting of specific charts, or when you have a bug in your graph and you would like to reproduce your code quickly to fix the bug, we recommend the usage of [Plotly's built-in datasets](#).

## 2 Layout and Styling

---

### 2.1 Layout: How to Line Things Up

A web app dashboard is displayed on a web browser as a webpage, which is written in HTML. The Div's role is to describe sections of an HTML document so the web browser knows exactly where to position each app component on the webpage. Consequently, every component used in the app – Dropdown, Graph, Header, Link, etc. – will be contained inside the initial Div.

Assume we create a web app with a layout of 3 Dropdowns inside 3 separate Divs:

```
app.layout = html.Div([
    html.Div(dcc.Dropdown()),
    html.Div(dcc.Dropdown()),
    html.Div(dcc.Dropdown()),
])
```

Imagine the webpage layout consisting of rows and columns. Because we did not define any row or column in the app layout above, the first Dropdown will appear on the top of the page and will fill the whole page from left to right. The second Dropdown will appear right below the first Dropdown and fill the whole width of the page as well. And so on with the third Dropdown. In other words, each Div represents a full row on the webpage, from top to bottom, unless we specifically define rows and columns in the layout.

Fortunately, there is a quick and easy way to define the rows and columns in Dash: It's done through a **CSS** stylesheet and a Div **className**. To use a CSS stylesheet, we must find one and tell our app to incorporate it. Row 20 of the `twitter_app.py` file grabs a custom CSS stylesheet from the web, and line 21 incorporates it into the app:

```
stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=stylesheets)
twitter_app.py file: rows 20 – 21
```

We use `dash.Dash()` to initiate the app. The `__name__` parameter inside the parenthesis helps the app find the assets folder. Our current web app dashboard does not have an assets folder, but sometimes the CSS stylesheet will be inside the assets



folder, which is why this `__name__` parameter is needed. We will go over the assets folder at greater length in Part III.

Once incorporated, we can use the `className` keyword argument of the `html.Div()` to refer to the CSS stylesheet and define the rows and columns of the app's components. First, we must define our rows because the columns should always be wrapped by rows. To do so, we set a string value "row" to the `className`. Let's build on the example in the previous page:

```
app.layout = html.Div([
    html.Div(dcc.Dropdown()),
    html.Div(dcc.Dropdown()),
    html.Div(dcc.Dropdown()),
], className="row")
```

After having defined the row, the next step is to define the columns of each `html.Div()` contained inside that row. To tell the app how many columns of space each component should fill, we set a string value to the `className`. The options are: "one column", "two columns", "three columns", [...] , "twelve columns". Given that the webpage consists of 12 columns and an infinite amount of rows, the sum of the components' column width must never surpass 12.

```
app.layout = html.Div([
    html.Div(dcc.Dropdown(), className="four columns"),
    html.Div(dcc.Dropdown(), className="four columns"),
    html.Div(dcc.Dropdown(), className="four columns"),
], className="row")
```

As you can see in the example above, the sum of all the app components is 12 columns. However, it doesn't always have to add up to 12. Going back to our web app dashboard, we see that the last row contains two `Divs`, with a sum of 5 columns only:

```
html.Div([

    html.Div(
        dcc.Dropdown(id="my-dropdown", multi=True,
                     options=[{'label':x, 'value':x}
                              for x in sorted(df['name'].unique())],
                     value=["taylorswift13", "cristiano", "jtimberlake"]),
        className="three columns"),

    html.Div(
        html.A(id="my-link", children="Click here to Visit Twitter",
               href="https://twitter.com/explore", target="_blank"),
        className="two columns")

], className="row"),
twitter_app.py file: rows 31 - 45
```

## 2.2 Styling: Embellish your App

Howeverlksajf sfd;lkdsjf lds;fj ds;lfkdsjf df;ldskjf ds;lfkjds fs.df dsalfkjdsf dslakfj  
dsfds;lafj dskdsjf sldfkjdsfldsakj fdsfsdf  
sdfdsf