**Dash** – Data visualisation notes from [Develop Data Visualization Interfaces in Python With Dash – Real Python](https://realpython.com/python-dash/)

<https://www.youtube.com/watch?v=UYGwgHhazMA>

Setting up

Create a folder for the web app including the dataset (csv file) and a virtual environment (call it venv/) to work in

Install dash and pandas in the virtual environment:

pip install dash pandas

Structure of building a dash application:

1. Define the content of the application (using app’s layout)
2. Style looks of app (using css)
3. Callbacks to determine interactive parts of the app and what they do
4. Use PythonAnywhere to deploy app online

Layout

Initialising dash application:

Create file app.py

Import relevant libraries:

import pandas as pd

from dash import Dash, dcc, html

Dash: initialises application

Dcc: allows creation of interactive components (graphs, dropdowns etc.)

html: allows access to HTML tags

Read and preprocess the data -> use pandas library (do separate notes and course on this)

Initialise the app using Dash library:

app = Dash(\_\_name\_\_)

Defining the layout of the dash application:

app initialised has a property called layout. To access it use:

app.layout

To create the layout use properties inside html module (same as the attributes <> in html)

To add graphs use properties inside dcc module

Text

Description automatically generated

Full list of dash HTML components: [Dash HTML Components | Dash for Python Documentation | Plotly](https://dash.plotly.com/dash-html-components)

List of dash core components provide abstractions for interactive interfaces: [Dash Core Components | Dash for Python Documentation | Plotly](https://dash.plotly.com/dash-core-components)

Dash uses plotly.js to generate graphs

Python dictionary is input to the dcc.Graph() attribute

To run the application:

if \_\_name\_\_ == “\_\_main\_\_”:

app.run\_server(debug=True)

These lines allow your Dash application to run locally using Flask’s built-in server

Go to <http://localhost:8050> to find the dashboard

Styling the dash application

Packages on PyPI ([Search results · PyPI](https://pypi.org/search/?q=dash%20components)) contain different styling features (dash-bootstrap-components is useful)

Applying custom style:

Two ways to style components:

Use style argument of individual items (using className and id)

Provide an external CSS file (saved within the same folder that app.py is in)

See notes about CSS for adding style attributes to an element

eg.

Graphical user interface

Description automatically generated with medium confidence

Adding external assets to your application:

Search how to add a favicon (image for tab in web browser) from [GitHub - twitter/twemoji: Emoji for everyone. https://twemoji.twitter.com/](https://github.com/twitter/twemoji)

Store all style files and external assets (eg. favicon) into a folder (inside the project folder) called assets/

Adding an external style sheet and title:

Graphical user interface, text, application

Description automatically generated

To a className into the app.py file:

Graphical user interface, text, application, chat or text message

Description automatically generated

Adding interactivity

Dash’s interactivity is based on a reactive programming paradigm

eg. user interacts with slider/dropdown then the output will automatically react

Creating interactive components:

No longer filter data when you initially read them. Instead you find the specific info (eg. regions and avocado\_types) that is present in the data:

Graphical user interface, text, application, chat or text message

Description automatically generated

Use dcc module to create a dropdown:

Text

Description automatically generated

options expects a dictionary with labels and values

clearable allows user to leave this field empty

Create a dcc.Graph however do not include the figure arguments because a callback function generates the figure argument using inputs the user sets

Defining Callbacks:

Dash’s callback functions are regular python functions with an app.callback decorator

When an input changes, callback function is triggered

Callback functions performs some predetermined operations (eg. filtering) and returns output to application



Text

Description automatically generated

Inside the app.callback decorator, you define the inputs and outputs.

The outputs take two arguments:

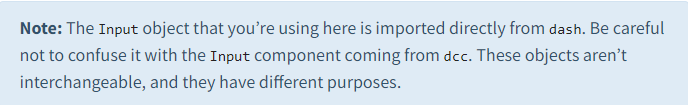
1. Identifier of the element that they’ll modify
2. Property of the element to be modified

eg. Output(“price-chart”, “figure”) will update the figure property of the price-chart element

The inputs take two arguments:

1. Identifier of the element that they’ll be watching for changes
2. Property of the watched element that they’ll be watching for changes

eg. Input(“region-filter”, “value”) will watch “region-filter” element and its value property for changes. Argument passed on to the callback function will be the new value of region-filter.value



Deploy the dash application to PythonAnywhere

s