I don’t know about you, but I occasionally find it a little bit intimidating to have to code something. This is doubly so when I’m building something akin to *web development* rather than doing some local data analysis and visualisation. I’m a competent Python coder, but I wouldn’t call myself a web developer at all, even after having more than dabbled with Django and Flask.

Still, converting your data outputs to a web app leads to a few non-trivial improvements for your project.

It is just much easier to build in true, powerful interactivity into a web app. It also means that you can control exactly how the data is presented, as the web app can become the de facto report as well as the access point to your data. Lastly, and most importantly, you can exponentially scale the accessibility to your outputs; making them available anywhere, any time. There is always a web browser at a user’s fingertips.

So, I bit the bullet and started to do just this with some of my data projects recently, with surprisingly fast speed and efficiency. I converted one of my outputs from [this article](https://towardsdatascience.com/create-effective-data-visualizations-of-proportions-94b69ad34410) to a [web app here](https://dash-nba-shot-dists.herokuapp.com) in just a couple of hours.

I thought this was rather cool, and wanted to share how this came together in just a few lines of code.

As always, I include everything you need to replicate my steps (data & code), and the article is not really about basketball. So do not worry if you are unfamiliar with it, and let’s get going.

# Before we get started

## Data

I include the code and data in my [GitLab repo here](https://gitlab.com/jphwang/online_articles) (dash\_simple\_nbadirectory). So please feel free to play with it / improve upon it.

## Packages

I assume you’re familiar with python. Even if you’re relatively new, this tutorial shouldn’t be too tricky, though.

You’ll need pandas, plotly and dash. Install each (in your virtual environment) with a simple pip install [PACKAGE\_NAME].

# Previously, on Python…

For this tutorial, I am simply going to skip \*most\* of the steps taken to create the local version of our visualisation. If you’re interested in what is going on, take a look at this article:

https://towardsdatascience.com/create-effective-data-visualizations-of-proportions-94b69ad34410

We will have a recap session, though, so you can see what is happening between plotting the chart locally with Plotly, and how to port that to a web app with Plotly Dash.

## Load data

I have pre-processed the data, and saved it as a CSV file. It is a collection of player data for the current NBA season (as of 26/Feb/2020), which shows:

* What share of their team’s shots they are taking, and
* How efficient / effective they are at doing it.

**For this portion, follow along by opening** local\_plot.py **in my repo.**

Load the data with:

all\_teams\_df = pd.read\_csv(‘srcdata/shot\_dist\_compiled\_data\_2019\_20.csv’)

Inspect the data with all\_teams\_df.head(), and you should see:

Each player’s data has been compiled for each minute of the game (excluding overtime), with stats pl\_acc and pl\_pps being the only exception, as they have been compiled per quarter of the game (for each 12 minute period).

The dataframe contains all NBA players, so let’s break it down to a manageable size, by filtering for a team. For instance, the New Orleans Pelicans’ players can be chosen with:

all\_teams\_df[all\_teams\_df.group == 'NOP']

Then, our data can be visualised in Plotly, as below:

import plotly.express as px  
fig = px.scatter(all\_teams\_df[all\_teams\_df.group == 'NOP'], x='min\_mid', y='player', size='shots\_freq', color='pl\_pps')  
fig.show()

I do add a few small details to my chart, to produce this version of the same graph.

This is the code that I used to do it.

Now, while it’s a lot of formatting code, I thought it useful to show you how I did it, because we are going to be re-using these functions in our Dash version of the code.

|  |
| --- |
| def clean\_chart\_format(fig): |
|  |

|  |
| --- |
| import plotly.graph\_objects as go |
|  |

|  |
| --- |
| fig.update\_layout( |
|  |

|  |
| --- |
| paper\_bgcolor="white", |
|  |

|  |
| --- |
| plot\_bgcolor="white", |
|  |

|  |
| --- |
| annotations=[ |
|  |

|  |
| --- |
| go.layout.Annotation( |
|  |

|  |
| --- |
| x=0.9, |
|  |

|  |
| --- |
| y=1.02, |
|  |

|  |
| --- |
| showarrow=False, |
|  |

|  |
| --- |
| text="Twitter: @\_jphwang", |
|  |

|  |
| --- |
| xref="paper", |
|  |

|  |
| --- |
| yref="paper", |
|  |

|  |
| --- |
| textangle=0 |
|  |

|  |
| --- |
| ), |
|  |

|  |
| --- |
| ], |
|  |

|  |
| --- |
| font=dict( |
|  |

|  |
| --- |
| family="Arial, Tahoma, Helvetica", |
|  |

|  |
| --- |
| size=10, |
|  |

|  |
| --- |
| color="#404040" |
|  |

|  |
| --- |
| ), |
|  |

|  |
| --- |
| margin=dict( |
|  |

|  |
| --- |
| t=20 |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
| fig.update\_traces(marker=dict(line=dict(width=1, color='Navy')), |
|  |

|  |
| --- |
| selector=dict(mode='markers')) |
|  |

|  |
| --- |
| fig.update\_coloraxes( |
|  |

|  |
| --- |
| colorbar=dict( |
|  |

|  |
| --- |
| thicknessmode="pixels", thickness=15, |
|  |

|  |
| --- |
| outlinewidth=1, |
|  |

|  |
| --- |
| outlinecolor='#909090', |
|  |

|  |
| --- |
| lenmode="pixels", len=300, |
|  |

|  |
| --- |
| yanchor="top", |
|  |

|  |
| --- |
| y=1, |
|  |

|  |
| --- |
| )) |
|  |

|  |
| --- |
| fig.update\_yaxes(showgrid=True, gridwidth=1, tickson='boundaries', gridcolor='LightGray', fixedrange=True) |
|  |

|  |
| --- |
| fig.update\_xaxes(showgrid=True, gridwidth=1, gridcolor='LightGray', fixedrange=True) |
|  |

|  |
| --- |
| return True |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def make\_shot\_dist\_chart(input\_df, color\_continuous\_scale=None, size\_col='shots\_count', col\_col='pl\_acc', range\_color=None): |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| max\_bubble\_size = 15 |
|  |

|  |
| --- |
| if color\_continuous\_scale is None: |
|  |

|  |
| --- |
| color\_continuous\_scale = px.colors.diverging.RdYlBu\_r |
|  |

|  |
| --- |
| if range\_color is None: |
|  |

|  |
| --- |
| range\_color = [min(input\_df[col\_col]), max(input\_df[col\_col])] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| fig = px.scatter( |
|  |

|  |
| --- |
| input\_df, x='min\_mid', y='player', size=size\_col, |
|  |

|  |
| --- |
| color=col\_col, |
|  |

|  |
| --- |
| color\_continuous\_scale=color\_continuous\_scale, |
|  |

|  |
| --- |
| range\_color=range\_color, |
|  |

|  |
| --- |
| range\_x=[0, 49], |
|  |

|  |
| --- |
| range\_y=[-1, len(input\_df.player.unique())], |
|  |

|  |
| --- |
| hover\_name='player', hover\_data=['min\_start', 'min\_end', 'shots\_count', 'shots\_made', 'shots\_freq', 'shots\_acc', ], |
|  |

|  |
| --- |
| render\_mode='svg' |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
| fig.update\_coloraxes(colorbar=dict(title='Points per<BR>100 shots')) |
|  |

|  |
| --- |
| fig.update\_traces(marker=dict(sizeref=2. \* 30 / (max\_bubble\_size \*\* 2))) |
|  |

|  |
| --- |
| fig.update\_yaxes(title="Player") |
|  |

|  |
| --- |
| fig.update\_xaxes(title='Minute', tickvals=list(range(0, 54, 6))) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return fig |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| fig = make\_shot\_dist\_chart( |
|  |

|  |
| --- |
| all\_teams\_df[all\_teams\_df.group == 'SAS'], col\_col='pl\_pps', range\_color=[90, 120], size\_col='shots\_freq') |
|  |

|  |
| --- |
| clean\_chart\_format(fig) |
|  |

|  |
| --- |
| fig.update\_layout(height=500, width=1250) |
|  |

fig.show()

Now, let’s get to the main event — how to create a web app out of these plots.

# Into the World Wide Web

You can read more about Plotly Dash [here](https://dash.plot.ly/introduction), but for now all you need to know that it is an open-source software package developed to abstract away the difficulties in putting your visualisations on the web.

It works with Flask under the hood, and you can happily reuse most of the code that you used to develop plots in plotly.py.

This is the simple version that I put together:

|  |
| --- |
| import pandas as pd |
|  |

|  |
| --- |
| import dash |
|  |

|  |
| --- |
| import dash\_core\_components as dcc |
|  |

|  |
| --- |
| import dash\_html\_components as html |
|  |

|  |
| --- |
| from dash.dependencies import Input, Output |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| all\_teams\_df = pd.read\_csv('srcdata/shot\_dist\_compiled\_data\_2019\_20.csv') |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| app = dash.Dash(\_\_name\_\_) |
|  |

|  |
| --- |
| server = app.server |
|  |

|  |
| --- |
| team\_names = all\_teams\_df.group.unique() |
|  |

|  |
| --- |
| team\_names.sort() |
|  |

|  |
| --- |
| app.layout = html.Div([ |
|  |

|  |
| --- |
| html.Div([dcc.Dropdown(id='group-select', options=[{'label': i, 'value': i} for i in team\_names], |
|  |

|  |
| --- |
| value='TOR', style={'width': '140px'})]), |
|  |

|  |
| --- |
| dcc.Graph('shot-dist-graph', config={'displayModeBar': False})]) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| @app.callback( |
|  |

|  |
| --- |
| Output('shot-dist-graph', 'figure'), |
|  |

|  |
| --- |
| [Input('group-select', 'value')] |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
| def update\_graph(grpname): |
|  |

|  |
| --- |
| import plotly.express as px |
|  |

|  |
| --- |
| return px.scatter(all\_teams\_df[all\_teams\_df.group == grpname], x='min\_mid', y='player', size='shots\_freq', color='pl\_pps') |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_': |
|  |

app.run\_server(debug=False)

What’s the big deal? Well, for one — it is a live web app, in under 25 lines of code. And notice the drop-down menu on the top left? Try changing the values on it, and watch the graph change \*magically\*.

Go on, I’ll wait.

Okay? Done.

Let’s briefly go through the code.

At a high level, what I’m doing here is to:

* Initialise a Dash app;
* Get a list of available team names, and provide it to a dropdown menu (with DOM id group-select) with a default value or ‘TOR’;
* Instantiate a Graph object as the shot-dist-graph identifier within Dash; and
* Create a callback function where if any of the values are changed, it will call the update\_graph function and pass the returned object to the Output.

If you take a look at the code, so many of what is probably trivial for web devs but annoying for me is abstracted away.

dcc.Graph wraps the figure object from plotly.py into my web app and HTML components like divs can be called and set up conveniently with html.Div objects.

Most gratifying for me personally is that Input objects and callbacks from those inputs are declaratively set up, and I can avoid having to deal with things like HTML forms or JavaScript.

And the resulting app still works beautifully. The graph is updated the moment that the pulldown menu is used to select another value.

And we did all that in fewer than 25 lines of code.

## Why Dash?

At this point, you might be asking — why Dash? We can do all this with a JS framework front end, and Flask, or any one of myriad other combinations.

To someone like me who prefers the comfort of Python than natively dealing with HTML and CSS, using Dash abstracts away a lot of stuff that doesn’t add a lot of value to the end product.

Take, for instance, a version of this app that includes further formatting and notes for the audience:

(It is simple\_dash\_w\_format.py in the git repo)

What’s the big deal? Well, for one — it is a live web app, in under 25 lines of code. And notice the drop-down menu on the top left? Try changing the values on it, and watch the graph change \*magically\*.

Go on, I’ll wait.

Okay? Done.

Let’s briefly go through the code.

At a high level, what I’m doing here is to:

* Initialise a Dash app;
* Get a list of available team names, and provide it to a dropdown menu (with DOM id group-select) with a default value or ‘TOR’;
* Instantiate a Graph object as the shot-dist-graph identifier within Dash; and
* Create a callback function where if any of the values are changed, it will call the update\_graph function and pass the returned object to the Output.

If you take a look at the code, so many of what is probably trivial for web devs but annoying for me is abstracted away.

dcc.Graph wraps the figure object from plotly.py into my web app and HTML components like divs can be called and set up conveniently with html.Div objects.

Most gratifying for me personally is that Input objects and callbacks from those inputs are declaratively set up, and I can avoid having to deal with things like HTML forms or JavaScript.

And the resulting app still works beautifully. The graph is updated the moment that the pulldown menu is used to select another value.

And we did all that in fewer than 25 lines of code.

**Why Dash?**

At this point, you might be asking — why Dash? We can do all this with a JS framework front end, and Flask, or any one of myriad other combinations.

To someone like me who prefers the comfort of Python than natively dealing with HTML and CSS, using Dash abstracts away a lot of stuff that doesn’t add a lot of value to the end product.

Take, for instance, a version of this app that includes further formatting and notes for the audience:

(It is simple\_dash\_w\_format.py in the git repo)

|  |
| --- |
| def clean\_chart\_format(fig): |
|  |

|  |
| --- |
| fig.update\_layout( |
|  |

|  |
| --- |
| paper\_bgcolor="white", |
|  |

|  |
| --- |
| plot\_bgcolor="white", |
|  |

|  |
| --- |
| annotations=[ |
|  |

|  |
| --- |
| go.layout.Annotation( |
|  |

|  |
| --- |
| x=0.9, |
|  |

|  |
| --- |
| y=1.02, |
|  |

|  |
| --- |
| showarrow=False, |
|  |

|  |
| --- |
| text="Twitter: @\_jphwang", |
|  |

|  |
| --- |
| xref="paper", |
|  |

|  |
| --- |
| yref="paper", |
|  |

|  |
| --- |
| textangle=0 |
|  |

|  |
| --- |
| ), |
|  |

|  |
| --- |
| ], |
|  |

|  |
| --- |
| font=dict( |
|  |

|  |
| --- |
| family="Arial, Tahoma, Helvetica", |
|  |

|  |
| --- |
| size=10, |
|  |

|  |
| --- |
| color="#404040" |
|  |

|  |
| --- |
| ), |
|  |

|  |
| --- |
| margin=dict( |
|  |

|  |
| --- |
| t=20 |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
| fig.update\_traces(marker=dict(line=dict(width=1, color='Navy')), |
|  |

|  |
| --- |
| selector=dict(mode='markers')) |
|  |

|  |
| --- |
| fig.update\_coloraxes( |
|  |

|  |
| --- |
| colorbar=dict( |
|  |

|  |
| --- |
| thicknessmode="pixels", thickness=15, |
|  |

|  |
| --- |
| outlinewidth=1, |
|  |

|  |
| --- |
| outlinecolor='#909090', |
|  |

|  |
| --- |
| lenmode="pixels", len=300, |
|  |

|  |
| --- |
| yanchor="top", |
|  |

|  |
| --- |
| y=1, |
|  |

|  |
| --- |
| )) |
|  |

|  |
| --- |
| fig.update\_yaxes(showgrid=True, gridwidth=1, tickson='boundaries', gridcolor='LightGray', fixedrange=True) |
|  |

|  |
| --- |
| fig.update\_xaxes(showgrid=True, gridwidth=1, gridcolor='LightGray', fixedrange=True) |
|  |

|  |
| --- |
| return True |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def make\_shot\_dist\_chart(input\_df, color\_continuous\_scale=None, size\_col='shots\_count', col\_col='pl\_acc', range\_color=None): |
|  |

|  |
| --- |
| max\_bubble\_size = 15 |
|  |

|  |
| --- |
| if color\_continuous\_scale is None: |
|  |

|  |
| --- |
| color\_continuous\_scale = px.colors.diverging.RdYlBu\_r |
|  |

|  |
| --- |
| if range\_color is None: |
|  |

|  |
| --- |
| range\_color = [min(input\_df[col\_col]), max(input\_df[col\_col])] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| fig = px.scatter( |
|  |

|  |
| --- |
| input\_df, x='min\_mid', y='player', size=size\_col, |
|  |

|  |
| --- |
| color=col\_col, |
|  |

|  |
| --- |
| color\_continuous\_scale=color\_continuous\_scale, |
|  |

|  |
| --- |
| range\_color=range\_color, |
|  |

|  |
| --- |
| range\_x=[0, 49], |
|  |

|  |
| --- |
| range\_y=[-1, len(input\_df.player.unique())], |
|  |

|  |
| --- |
| hover\_name='player', hover\_data=['min\_start', 'min\_end', 'shots\_count', 'shots\_made', 'shots\_freq', 'shots\_acc', ], |
|  |

|  |
| --- |
| render\_mode='svg' |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
| fig.update\_coloraxes(colorbar=dict(title='Points per<BR>100 shots')) |
|  |

|  |
| --- |
| fig.update\_traces(marker=dict(sizeref=2. \* 30 / (max\_bubble\_size \*\* 2))) |
|  |

|  |
| --- |
| fig.update\_yaxes(title="Player") |
|  |

|  |
| --- |
| fig.update\_xaxes(title='Minute', tickvals=list(range(0, 54, 6))) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return fig |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| app.title = 'Dash Demo - NBA' |
|  |

|  |
| --- |
| team\_names = all\_teams\_df.group.unique() |
|  |

|  |
| --- |
| team\_names.sort() |
|  |

|  |
| --- |
| app.layout = html.Div([ |
|  |

|  |
| --- |
| html.Div([ |
|  |

|  |
| --- |
| dcc.Markdown( |
|  |

|  |
| --- |
| """ |
|  |

|  |
| --- |
| #### Shot Frequencies & Efficiencies (2019-20 NBA Season) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| This page compares players based on shot \*frequency\* and \*efficiency\*, |
|  |

|  |
| --- |
| divided up into minutes of regulation time for each team. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Use the pulldown to select a team, or select 'Leaders' to see leaders from each team. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| \*Notes\*: |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| \* \*\*Frequency\*\*: A team's shots a player is taking, indicated by \*\*size\*\*. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| \* \*\*Efficiency\*\*: Points scored per 100 shots, indicated by \*\*colour\*\* (red == better, blue == worse). |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| \* Players with <1% of team shots are shown under 'Others' |
|  |

|  |
| --- |
| """ |
|  |

|  |
| --- |
| ), |
|  |

|  |
| --- |
| html.P([html.Small("See more data / NBA analytics content, find me on "), html.A(html.Small("twitter"), href="https://twitter.com/\_jphwang", title="twitter"), html.Small("!")]), |
|  |

|  |
| --- |
| ]), |
|  |

|  |
| --- |
| html.Div([ |
|  |

|  |
| --- |
| dcc.Dropdown( |
|  |

|  |
| --- |
| id='group-select', |
|  |

|  |
| --- |
| options=[{'label': i, 'value': i} for i in team\_names], |
|  |

|  |
| --- |
| value='TOR', |
|  |

|  |
| --- |
| style={'width': '140px'} |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
| ]), |
|  |

|  |
| --- |
| dcc.Graph( |
|  |

|  |
| --- |
| 'shot-dist-graph', |
|  |

|  |
| --- |
| config={'displayModeBar': False} |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
| ]) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| @app.callback( |
|  |

|  |
| --- |
| Output('shot-dist-graph', 'figure'), |
|  |

|  |
| --- |
| [Input('group-select', 'value')] |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
| def update\_graph(grpname): |
|  |

|  |
| --- |
| fig = make\_shot\_dist\_chart( |
|  |

|  |
| --- |
| all\_teams\_df[all\_teams\_df.group == grpname], col\_col='pl\_pps', range\_color=[90, 120], size\_col='shots\_freq') |
|  |

|  |
| --- |
| clean\_chart\_format(fig) |
|  |

|  |
| --- |
| if len(grpname) > 3: |
|  |

|  |
| --- |
| fig.update\_layout(height=850, width=1250) |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| fig.update\_layout(height=500, width=1250) |
|  |

|  |
| --- |
|  |
|  |

return fig

Most of the changes are cosmetic, but I will note that here, I just write the body text in Markdown, and simply carry over my formatting functions from Plotly to be used in the formatting the graphs in Dash.

This saves me a tremendous amount of time between doing data analysis and visualisation to deployment to clients’ views.

All in all, from starting with my initial graph, I think it probably took less than an hour to deploy it to Heroku. Which is pretty amazing.

I will get into more advanced features of Dash, and actually doing some cool things with it functionality-wise, but I was very happy with this outcome in terms of ease and speed.

Try it out yourself — I think that you’d be very impressed. Next time, I plan to write about some really cool things you can do with Dash, and building truly interactive dashboards.

**Edit**: I’m starting a **Substack** for **all things Data and Visualisation** — it is going to be a way for me to engage directly with you. I would love for you to join me here.

I also [wrote a little about why](https://medium.com/@_jphwang/introducing-the-visual-in-the-noise-b70d2c8f9b63) I’m starting a Substack.

If you liked this, say 👋 / follow on [twitter](https://twitter.com/_jphwang), or follow for updates. This is the article that the data viz is based on: