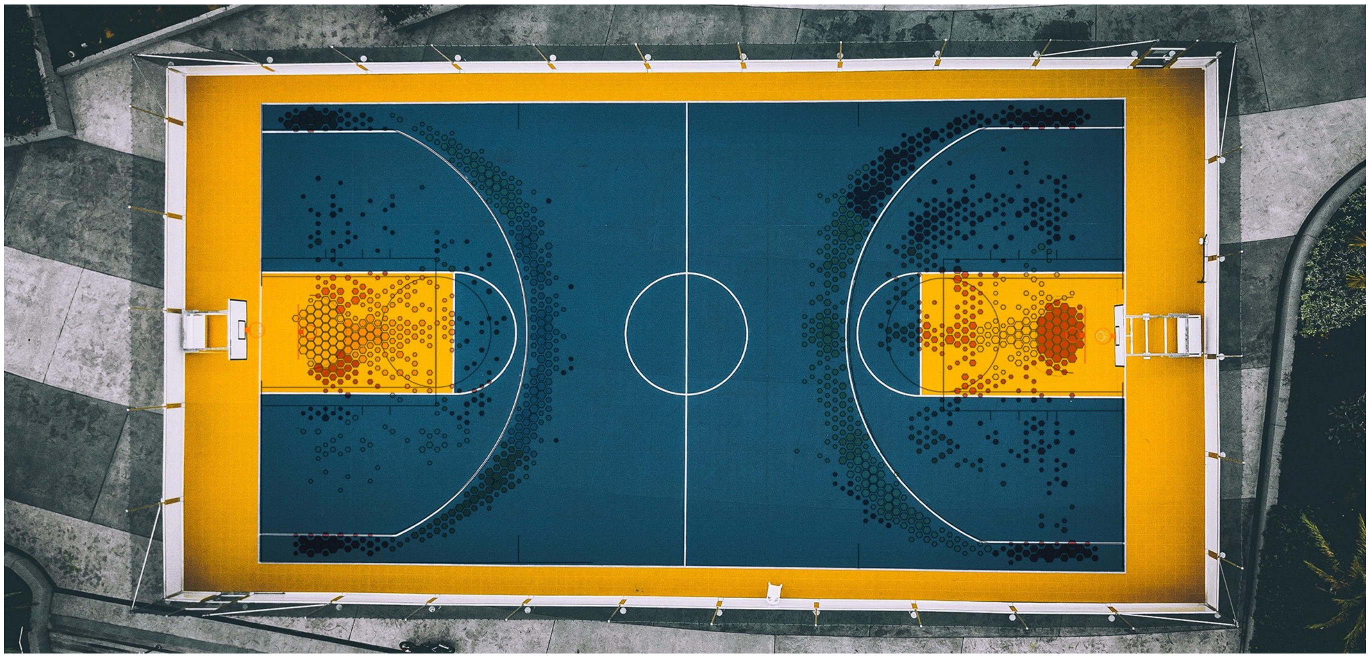
Interactive basketball data visualizations with Plotly

Analyze sports data with hexbin shot charts and bubble charts with Plotly and Plotly Express (source code & my own data for all 30 teams included in my GitLab repo)

JP Hwang Follow
Jan 27 · 11 min read ★



Original photo by Izuddin Helmi Adnan on Unsplash

> *This post is mostly about visualisation. It is only going to include very cursory basketball info — basically, if you know what it is, you're going to be fine.*

I have been tinkering with basketball data analysis & visualisation recently, using matplotlib for plotting. It is powerful, but Plotly is my usual visualisation package of choice.

In my opinion, Plotly achieves the right balance of power and customisability, written in sensible, intuitive syntax with great documentation and development rate.

So I recently migrated my basketball visualisation scripts to Plotly, with great results. In this article, I would like to share some of that, including examples using both Plotly and Plotly Express.

> *I included the code for this in my GitLab repo here (**basketball_plots** directory), so please feel free to download it and play with it / improve upon it.*

Before we get started

Data

As this article is mainly about visualization, I will include my pre-processed data outputs in my repo for all 30 teams & league average.

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 `plotly.` Install it (in your virtual environment) with a simple `pip install plotly`.

Bubble charts in a blink — with Plotly Express

Professional basketball players in the NBA take shots from right at the rim, to past the three-point line, which is about 24 feet away from it. I wanted to understand how the distance affects accuracy of shots, and how often players shoot from each distance, and see if there is a pattern.

This is a relatively straightforward exercise, so let's use Plotly Express. Plotly Express is a fairly new package, and is all about producing charts more quickly and efficiently, so you can focus on the data exploration. (You can read more about it here)

I have a database of all shot locations for an entire season (2018–2019 season) of shots, which is about 220,000 shots. The database includes locations of each shot, and whether it was successful (made) or not.

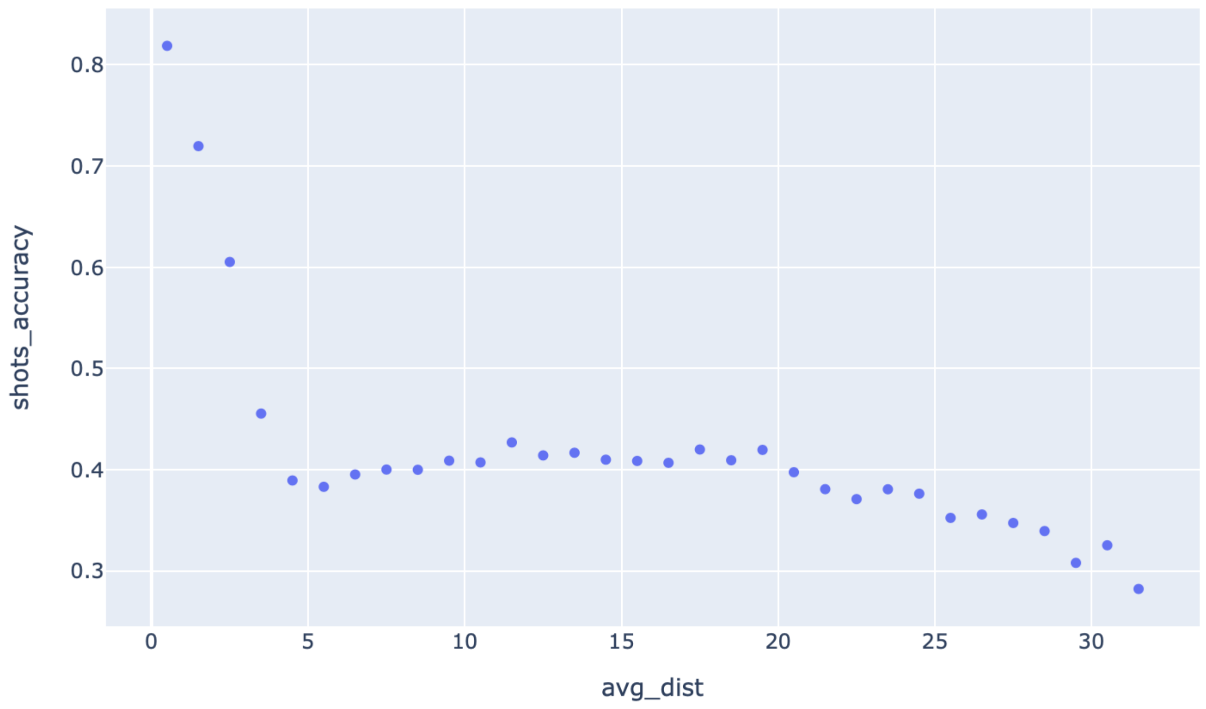
Using this data, I produced a summary (`league_shots_by_dist.csv`), which includes shots grouped by distance in 1-foot bins (up to 32 feet), and columns for `shots_made`, `shots_counts`, `shots_acc` and `avg_dist`.

So let's explore this data — simply load the data with:

```
import pandas as pd
grouped_shots_df = pd.read_csv('srcdata/league_shots_by_dist.csv')
```

And then after importing the package, running just the two lines of code below will magically open an interactive scatter chart on your browser.

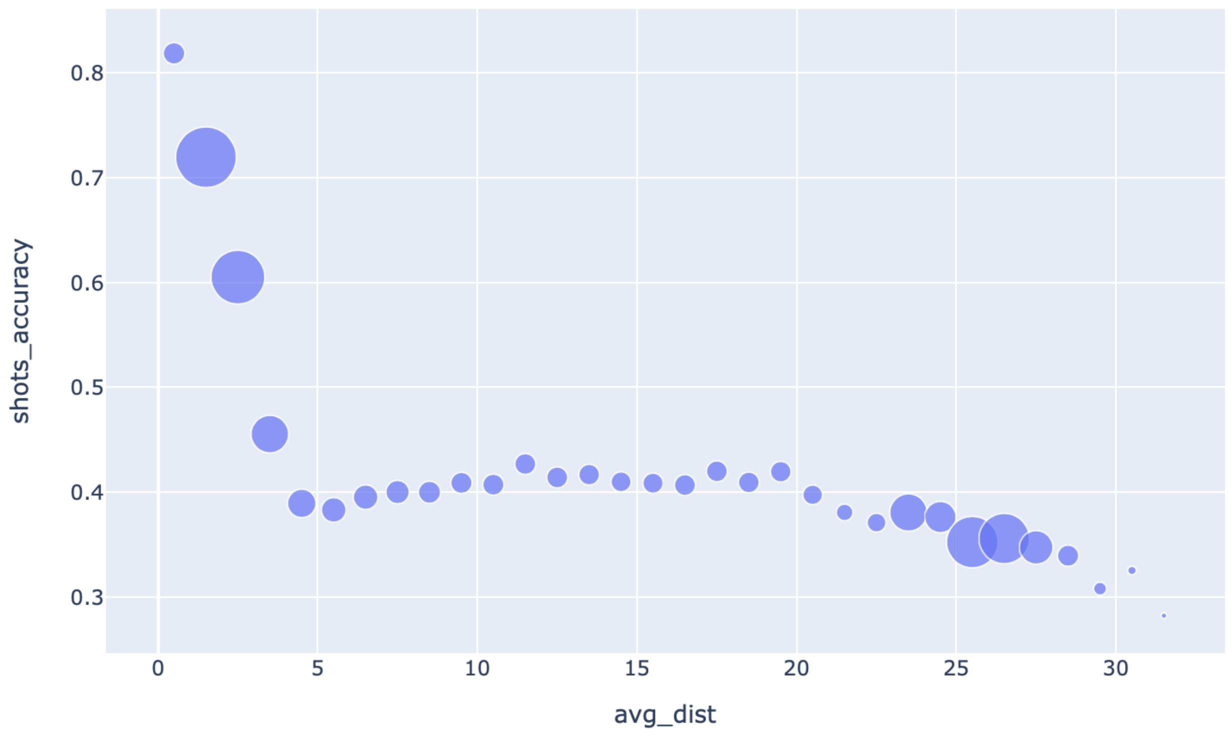
```
import plotly.express as px
fig = px.scatter(grouped_shots_df, x="avg_dist", y="shots_accuracy")
fig.show()
```



Simple scatter plot, in just two lines of code

To see how often players shoot from each distance, let's add the frequency data: simply pass `shots_counts` value to the `'size'` parameter, and specify a maximum bubble size.

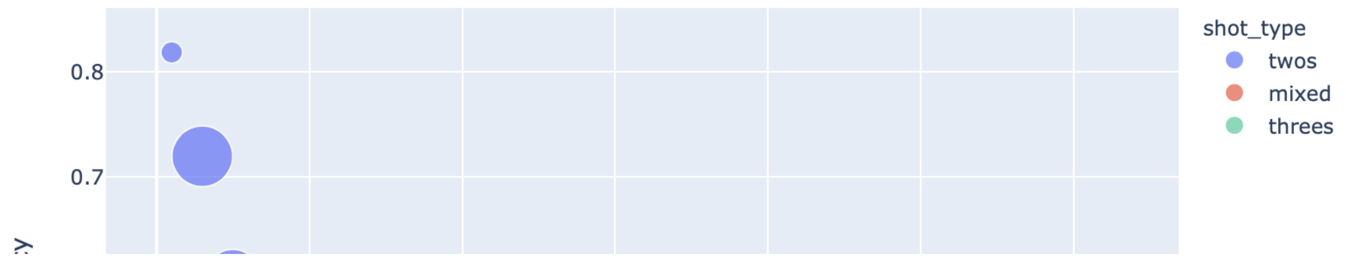
```
fig = px.scatter(grouped_shots_df, x="avg_dist", y="shots_accuracy",
size="shots_counts", size_max=25)
fig.show()
```

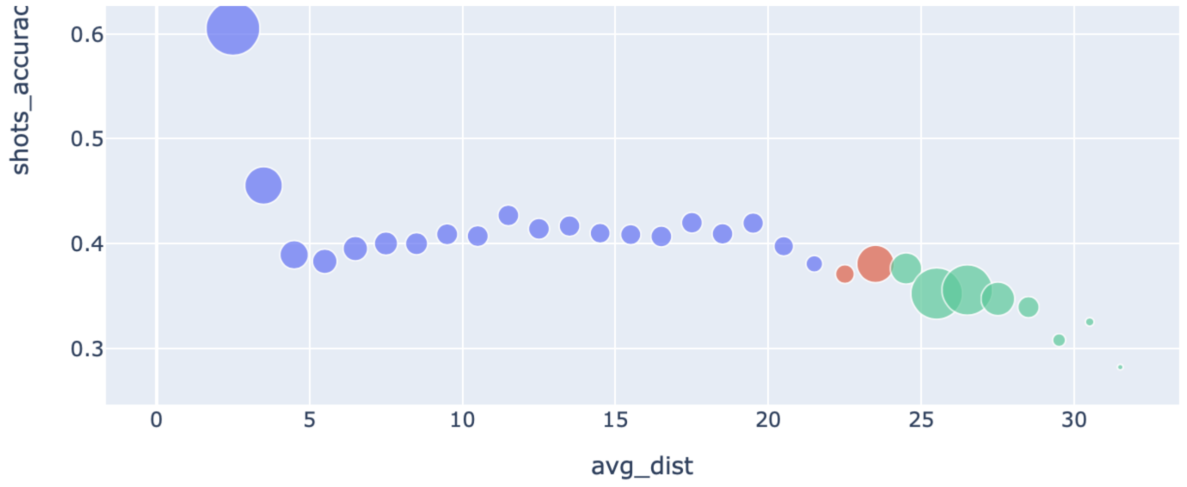


Simple bubble chart, still in just two lines of code

That's intersting. The frequency (bubble size) decreases, and then picks back up again. Why is that? Well, we know that as we get farther, some of these are two point shots, some are three pointers, and some are a mix of the two. So let's try colouring the variables by the shot type.

```
fig = px.scatter(grouped_shots_df, x="avg_dist", y="shots_accuracy",
size="shots_counts", color='shot_type', size_max=25)
fig.show()
```



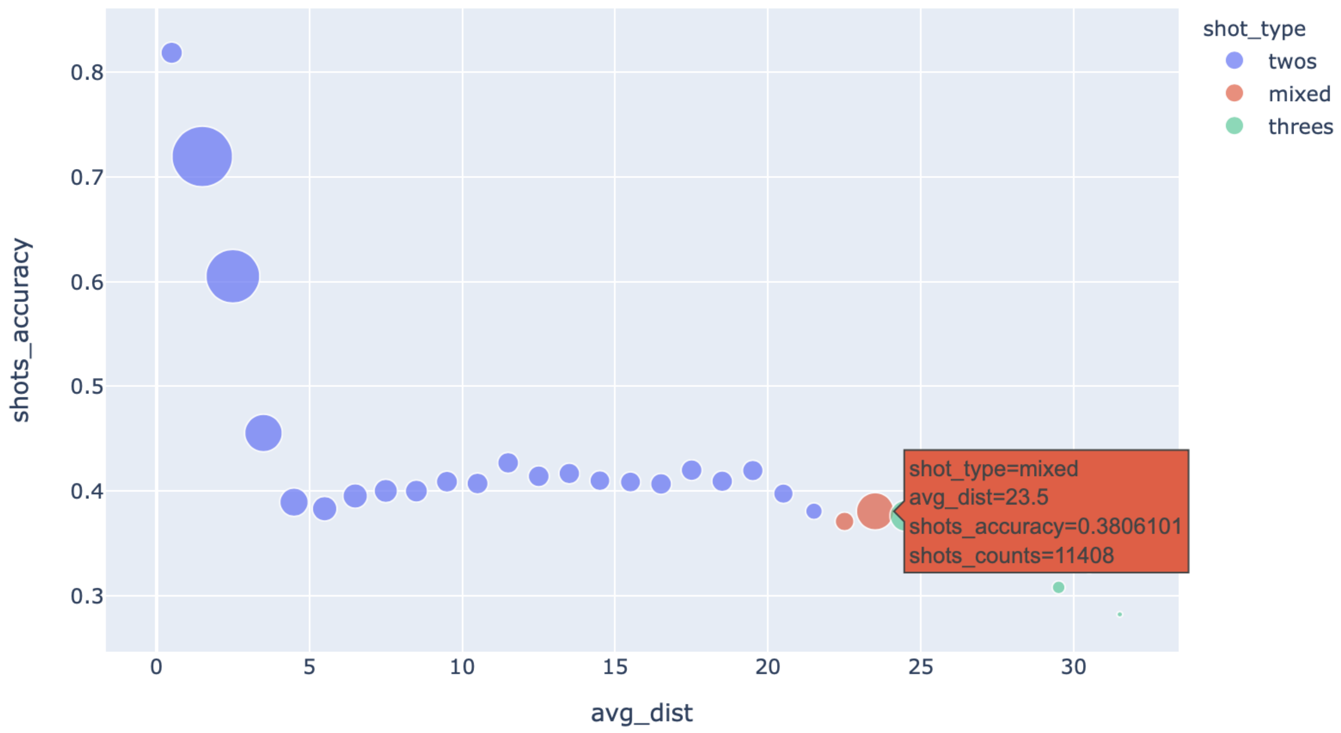


Bubble chart, with categories — **still** in just two lines of code!

Ah, there it is. It looks like the shot frequency increases as players try to take advantage of the three point line.

> *Edit: here's a live demo*

Try moving your mouse over each point — you will be pleasantly rewarded with a text tooltip! You didn't even have to set anything up.



Default mouseover tooltips for details

Moving your cursor and looking at the individual points, the data tells us that shot accuracy doesn't change greatly past 5 to 10 feet from the basket. By shooting threes, the players are trading off about a 15% decrease in shot accuracy for a 50% more reward of a three pointer. It makes sense that three pointers are more popular than these 'mid-range' two pointers.

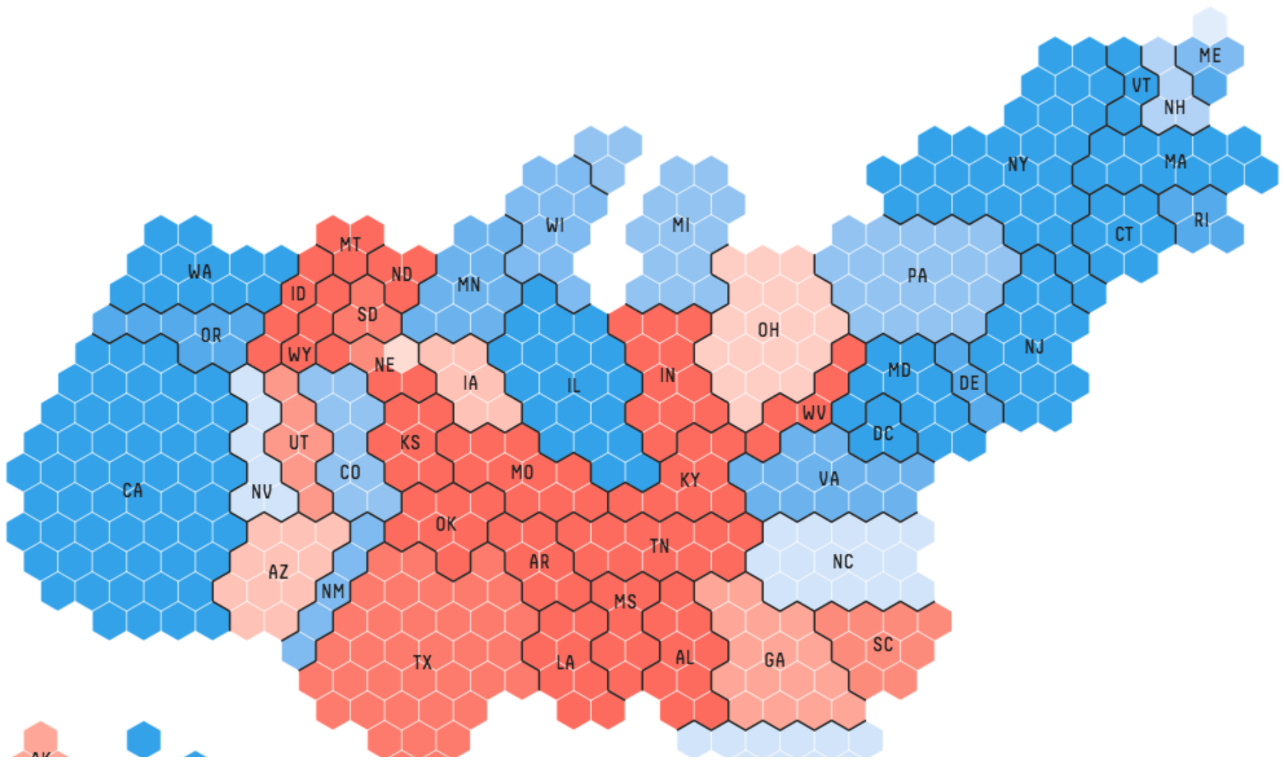
That's not exactly a groundbreaking conclusion, but it's still neat to be able to see it for ourselves.

But more importantly, wasn't that insanely simple? **We created the last chart with just two lines of code!**

As long as you have a 'tidy' dateframe that has been pre-processed, Plotly Express allows fast visualisations like this, which you can work from. It's a fantastic tool for data exploration.

Hexbin plots, with Plotly

Let's move onto another chart, called hexbin charts. I've discussed it elsewhere, but hexbin charts allow area-based visualisation of data, by dividing an entire area into hexagon-sized grids and displaying data by their colour (and also sometimes size) like so.



Hexbins in election coverage (Information is beautiful / fivethirtyeight)

While Plotly does not natively provide functions to compile hexbin data from coordinate-based data points, it does not matter for us because a) `matplotlib` does (read about the *`Polycollection`* that is returned by `matplotlib` here, if you are interested), and b) I will be providing the dataset for use here.

Okay, so let's move straight onto visualisation of the hexbin data:

Our first shot chart

I have saved the data in a dictionary format. Simply load the data with:

```
import pickle
with open('srcdata/league_hexbin_stats.pickle', 'rb') as f:
    league_hexbin_stats = pickle.load(f)
```

The dictionary contains these keys (see for yourself with `print(league_hexbin_stats.keys())`:

```
['xlocs', 'ylocs', 'shots_by_hex', 'freq_by_hex', 'accs_by_hex',
'shot_ev_by_hex', 'gridsize', 'n_shots']
```

The important ones are: x & y location data `xlocs`, `ylocs`, frequency data `freq_by_hex` and accuracy data `accs_by_hex`. Each of these include data from each hexagon, except for accuracy data, which I have averaged into "zones" to smooth out local variations. You'll know what I mean when you see the plots.

> *Note: The X/Y data are as captured originally, according to the standard coordinate system. I base everything else from it. Basically, the centre of the rim is at (0, 0) and 1 in X & Y coordinates appears to be 1/10th of a foot.*

Let's plot those. This time we will use `plotly.graph_objects`, for the extra flexibility it gives us. It leads to writing slightly longer code, but don't worry — it's still not very long, and it will be totally worth it.

```python
xlocs = league_hexbin_stats['xlocs']
ylocs = league_hexbin_stats['ylocs']
accs_by_hex = league_hexbin_stats['accs_by_hex']
freq_by_hex = league_hexbin_stats['freq_by_hex']

import plotly.graph_objects as go
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=xlocs, y=ylocs, mode='markers', name='markers',
    marker=dict(
        size=freq_by_hex, sizemode='area', sizeref=2. * max(freq_by_hex) / (11. ** 2), s
        color=accs_by_hex,
        line=dict(width=1, color='#333333'), symbol='hexagon',
    ),
))
fig.show(config=dict(displayModeBar=False))
```

basketball_plots_hexbin_plot1.txt hosted with ❤ by GitHub view raw

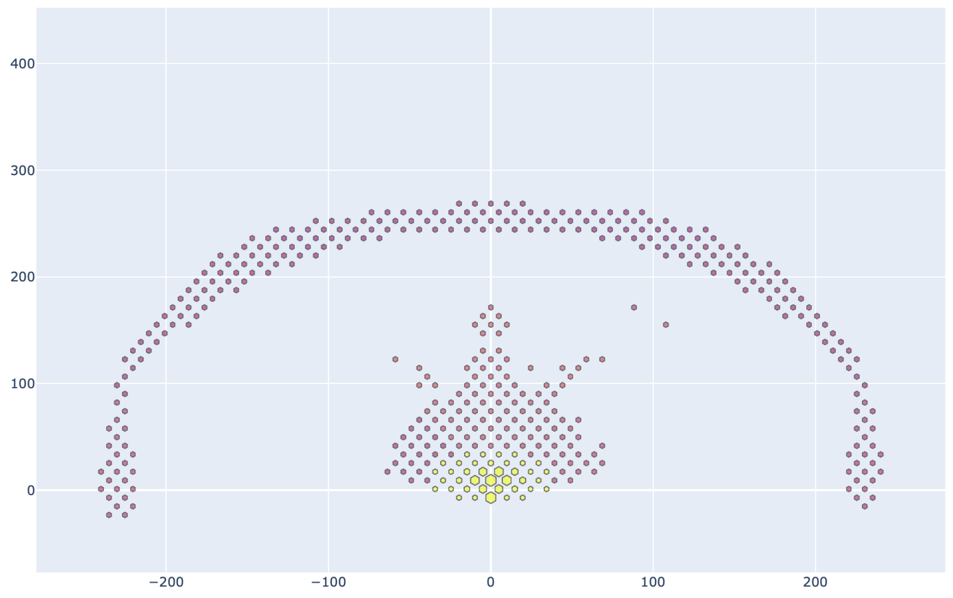
Let's go through this. The first few lines are obvious — I am just giving a few values new names, so that I can reuse the plotting code more easily. `go.Figure()` creates and returns a new figure, which we assign to `fig`.

Then we add a new scatter plot, with `markers` mode (i.e. no lines), and we specify parameters for those markers, including passing arrays/lists to be our sizes and colours.

The `sizeref` parameter gives a reference size to scale the rest of the sizes — I basically just play with this to get the right size, and `sizemode` refers to how the sizing works — whether the size should vary by area, or diameter.

The convention is that 'area' should be used, as changing the diameter according to the variable would change the area by the square of that value, and would exaggerate the differences.

We also specify the marker symbol as a `hexagon` (it is a hexbin plot, after all), and add a `line` on the outside of the hexagon for visual impact.



Our first Plotly shot chart (or abstract art, I'm not sure)

Looks… almost like a shot chart (or a message from our alien overlords), although obviously problematic. Where is the court, and why is the ratio funny? It's impossible to know what the colours mean. The mouseover tooltips just show me the X-Y coordinates, which is not very helpful.

Let's get to fixing those:

Draw me a picture

Luckily, Plotly provides a set of handy commands to draw whatever you want. By using the method `fig.update_figure`, and passing a list to the `shapes` parameter, it's pretty easy to draw whatever you would like.

I based the court dimensions on this handy wikipedia figure, and the court was created with a mix of rectangles, circles and lines. I won't go through the mundane details, but here are a couple of things you might find interesting:

- One thing that I had trouble figuring out was drawing SVG arcs, but this post on the plotly forum helped me out. It turns out plotly.js does not natively support arcs!

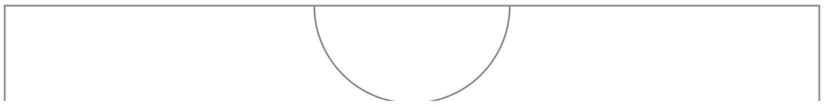
- Plotly allows me to fix the x & y ratio — I have disabled zoom here, but if it wasn't, you can fix the y-axis to the x-axis using: `yaxis=dict(scaleanchor="x", scaleratio=1)`.

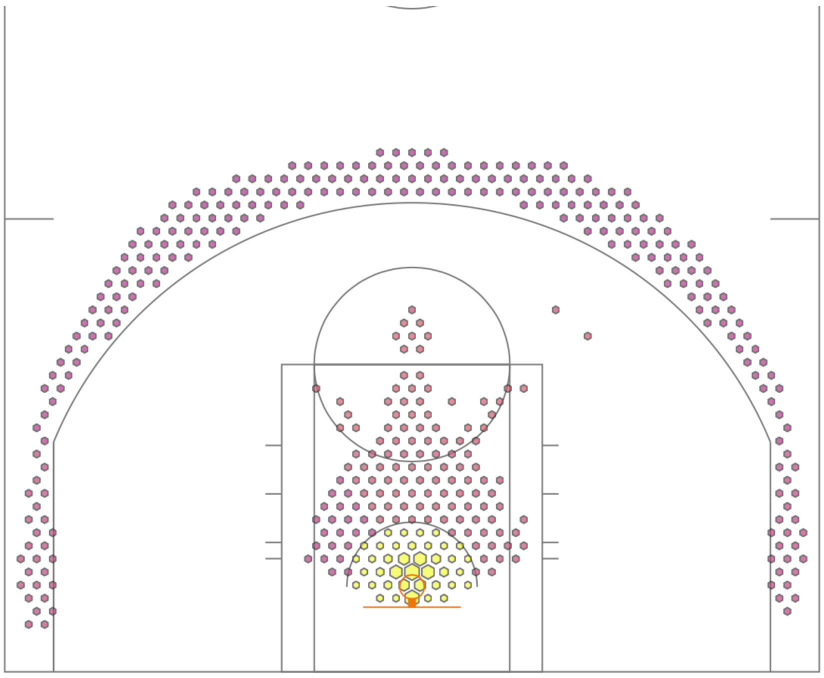
```
def draw_plotly_court(fig, fig_width=600, margins=10):

    import numpy as np

    # From: https://community.plot.ly/t/arc-shape-with-path/7205/5
    def ellipse_arc(x_center=0.0, y_center=0.0, a=10.5, b=10.5, start_angle=0.0, end_ar
        t = np.linspace(start_angle, end_angle, N)
        x = x_center + a * np.cos(t)
        y = y_center + b * np.sin(t)
        path = f'M {x[0]}, {y[0]}'
        for k in range(1, len(t)):
            path += f'L{x[k]}, {y[k]}'
        if closed:
            path += ' Z'
        return path

    fig_height = fig_width * (470 + 2 * margins) / (500 + 2 * margins)
    fig.update_layout(width=fig_width, height=fig_height)

    # Set axes ranges
    fig.update_xaxes(range=[-250 - margins, 250 + margins])
    fig.update_yaxes(range=[-52.5 - margins, 417.5 + margins])

    threept_break_y = 89.47765084
    three_line_col = "#777777"
    main_line_col = "#777777"

    fig.update_layout(
        # Line Horizontal
        margin=dict(l=20, r=20, t=20, b=20),
        paper_bgcolor="white",
        plot_bgcolor="white",
        yaxis=dict(
            scaleanchor="x",
            scaleratio=1,
            showgrid=False,
            zeroline=False,
            showline=False,
            ticks='',
            showticklabels=False,
            fixedrange=True,
        ),
        xaxis=dict(
            showgrid=False,
            zeroline=False,
            showline=False,
            ticks='',
            showticklabels=False,
            fixedrange=True,
        ),
        shapes=[
            dict(
                type="rect", x0=-250, y0=-52.5, x1=250, y1=417.5,
                line=dict(color=main_line_col, width=1),
                # fillcolor='#333333',
                layer='below'
            ),
            dict(
                type="rect", x0=-80, y0=-52.5, x1=80, y1=137.5,
                line=dict(color=main_line_col, width=1),
                # fillcolor='#333333',
                layer='below'
            ),
            dict(
                type="rect", x0=-60, y0=-52.5, x1=60, y1=137.5,
                line=dict(color=main_line_col, width=1),
                # fillcolor='#333333',
                layer='below'
            ),
            dict(
                type="circle", x0=-60, y0=77.5, x1=60, y1=197.5, xref="x", yref="y",
                line=dict(color=main_line_col, width=1),
                # fillcolor='#dddddd',
                layer='below'
            ),
            dict(
                type="line", x0=-60, y0=137.5, x1=60, y1=137.5,
                line=dict(color=main_line_col, width=1),
                layer='below'
            ),

            dict(
                type="rect", x0=-2, y0=-7.25, x1=2, y1=-12.5,
                line=dict(color="#ec7607", width=1),
                fillcolor='#ec7607',
            ),
            dict(
                type="circle", x0=-7.5, y0=-7.5, x1=7.5, y1=7.5, xref="x", yref="y",
                line=dict(color="#ec7607", width=1),
            ),
            dict(
                type="line", x0=-30, y0=-12.5, x1=30, y1=-12.5,
                line=dict(color="#ec7607", width=1),
            ),

            dict(type="path",
                 path=ellipse_arc(a=40, b=40, start_angle=0, end_angle=np.pi),
                 line=dict(color=main_line_col, width=1), layer='below'),
            dict(type="path",
                 path=ellipse_arc(a=237.5, b=237.5, start_angle=0.386283101, end_angle=
                 line=dict(color=main_line_col, width=1), layer='below'),
            dict(
                type="line", x0=-220, y0=-52.5, x1=-220, y1=threept_break_y,
                line=dict(color=three_line_col, width=1), layer='below'
            ),
            dict(
                type="line", x0=-220, y0=-52.5, x1=-220, y1=threept_break_y,
                line=dict(color=three_line_col, width=1), layer='below'
            ),
            dict(
                type="line", x0=220, y0=-52.5, x1=220, y1=threept_break_y,
                line=dict(color=three_line_col, width=1), layer='below'
            ),

            dict(
                type="line", x0=-250, y0=227.5, x1=-220, y1=227.5,
                line=dict(color=main_line_col, width=1), layer='below'
            ),
            dict(
                type="line", x0=250, y0=227.5, x1=220, y1=227.5,
                line=dict(color=main_line_col, width=1), layer='below'
            ),
            dict(
                type="line", x0=-90, y0=17.5, x1=-80, y1=17.5,
                line=dict(color=main_line_col, width=1), layer='below'
            ),
            dict(
                type="line", x0=-90, y0=27.5, x1=-80, y1=27.5,
                line=dict(color=main_line_col, width=1), layer='below'
            ),
            dict(
                type="line", x0=-90, y0=57.5, x1=-80, y1=57.5,
                line=dict(color=main_line_col, width=1), layer='below'
            ),
            dict(
                type="line", x0=-90, y0=87.5, x1=-80, y1=87.5,
                line=dict(color=main_line_col, width=1), layer='below'
            ),
            dict(
                type="line", x0=90, y0=17.5, x1=80, y1=17.5,
                line=dict(color=main_line_col, width=1), layer='below'
            ),
            dict(
                type="line", x0=90, y0=27.5, x1=80, y1=27.5,
                line=dict(color=main_line_col, width=1), layer='below'
            ),
            dict(
                type="line", x0=90, y0=57.5, x1=80, y1=57.5,
                line=dict(color=main_line_col, width=1), layer='below'
            ),
            dict(
                type="line", x0=90, y0=87.5, x1=80, y1=87.5,
                line=dict(color=main_line_col, width=1), layer='below'
            ),

            dict(type="path",
                 path=ellipse_arc(y_center=417.5, a=60, b=60, start_angle=-0, end_angle
                 line=dict(color=main_line_col, width=1), layer='below'),

        ]
    )
    return True
```

basketball_plots draw plotly court hosted with ❤ by **GitHub** view raw

draw_plotly_court function

Running the same command as above, but simply inserting `draw_plotly_court(fig)` between `fig = go.Figure(),` and `fig.add_trace(...`, we get:




draw_plotly_court in action

Putting a finger on the scale

Although much improved, we can't make much of the data. The colour scale , and the size scale are both not great.

To remedy these, we'll:

- 'Clip' the frequency values — by manually limiting values to my `max_freq` value, with a list comprehension.
- Introduce a different colour scale. I wanted to use a 'sequential' scale, as this shows changing positive values. This page shows all of the standard palettes that comes with Plotly. I chose '`YlOrRd`'.

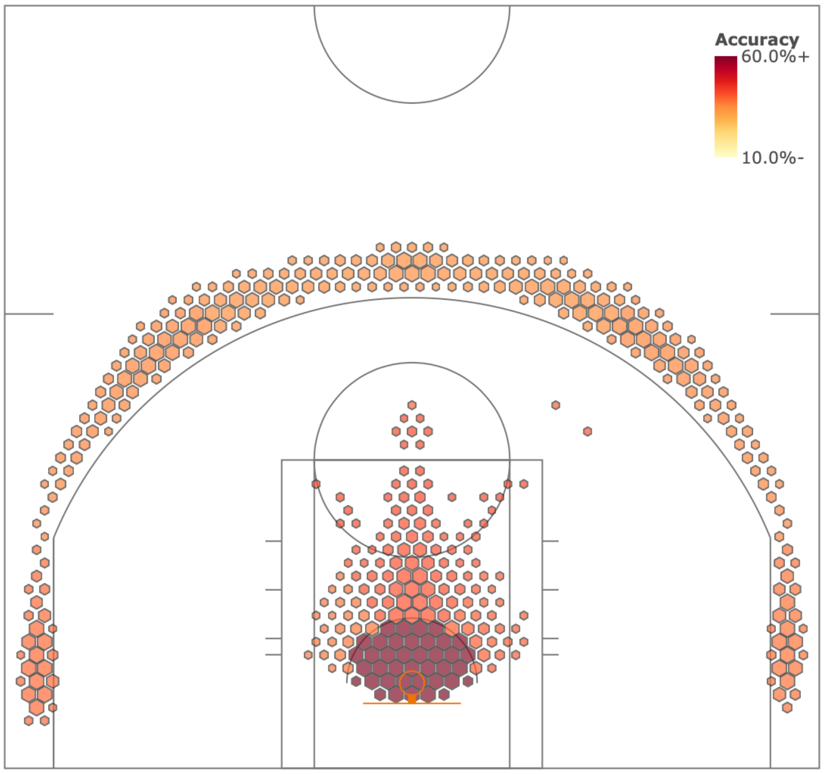
YlOrRd colour scale

- Specify a max/min value for the colours, and
- Add a legend on the plot. I chose to specify just the top/bottom tick values, and add the characters `+` and `-` to indicate that they are clipped.

Running the below code, you should be able to generate this chart.

```python
max_freq = 0.002
freq_by_hex = np.array([min(max_freq, i) for i in league_hexbin_stats['freq_by_hex']])
colorscale = 'YlOrRd'
marker_cmin = 0.1
marker_cmax = 0.6
ticktexts = [str(marker_cmin*100)+'%-', "", str(marker_cmax*100)+'%+']

fig = go.Figure()
draw_plotly_court(fig)
fig.add_trace(go.Scatter(
    x=xlocs, y=ylocs, mode='markers', name='markers',
    marker=dict(
        size=freq_by_hex, sizemode='area', sizeref=2. * max(freq_by_hex) / (11. ** 2), s
        color=accs_by_hex, colorscale=colorscale,
        colorbar=dict(
            thickness=15,
            x=0.84,
            y=0.87,
            yanchor='middle',
            len=0.2,
            title=dict(
                text="<B>Accuracy</B>",
                font=dict(
                    size=11,
                    color='#4d4d4d'
                ),
            ),
            tickvals=[marker_cmin, (marker_cmin + marker_cmax) / 2, marker_cmax],
            ticktext=ticktexts,
            tickfont=dict(
                size=11,
                color='#4d4d4d'
            )
        ),
        cmin=marker_cmin, cmax=marker_cmax,
        line=dict(width=1, color='#333333'), symbol='hexagon',
    ),
))
fig.show(config=dict(displayModeBar=False))
```

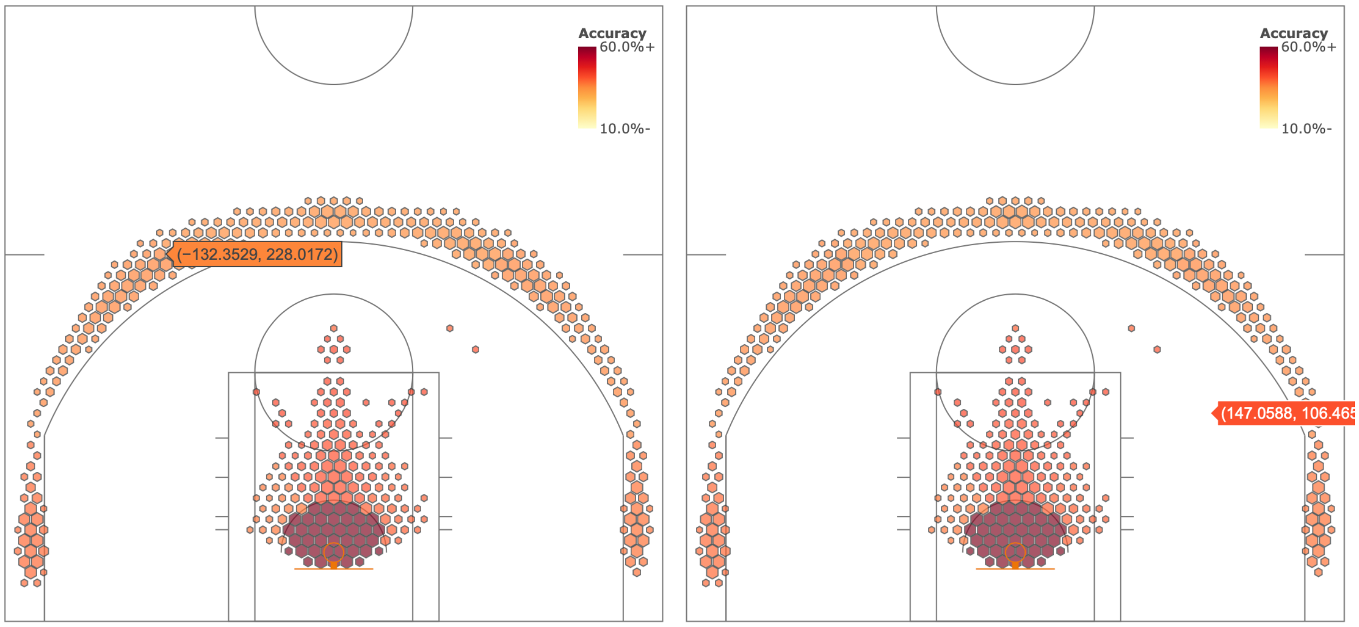
basketball_plots_hexbin_plot2.txt hosted with ❤ by GitHub view raw



Our shot chart, with a legend, and much better looking hexagons.

Tooltips

If you move your mouse over the chart, you'll tooltips even in areas where the frequency value in my hexbin data is zero (see bottom right image). Also, the tooltip data isn't currently very useful — it's just X&Y coordinates. What can we do?



Useless tooltips

First of all, let's just not have values at all where the frequencies are zero.

I wrote a simple function `filt_hexbins` to filter the hexbin data, so that if the frequency data is zero, it just deletes those values from all arrays of the same length.

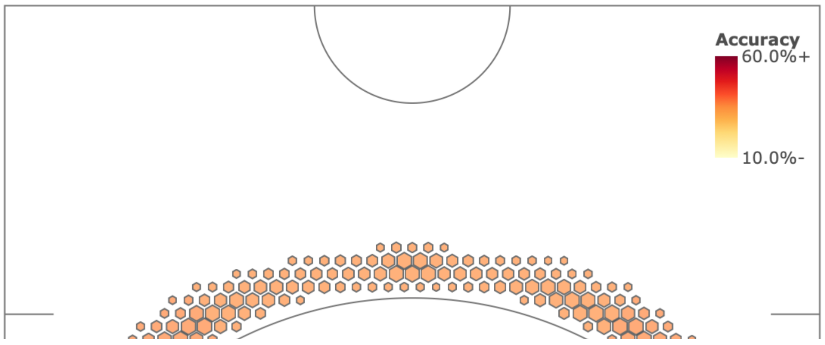
And for the tooltip, we simply pass a text list to the parameter `text`. Also change the `hoverinfo` parameter value to `text`, and you're good to go! The text string can include basic HTML like `<i>`, `<b>` or `<br>` tags, so try them out also.

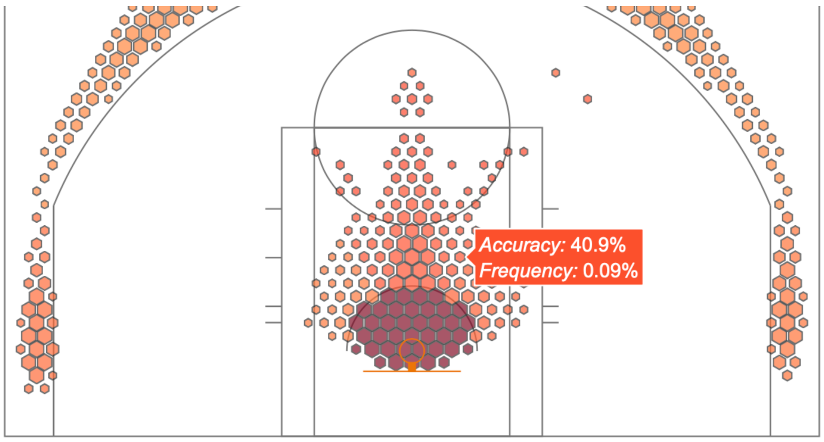
Putting it together, we get this code, and this chart, with the improved, informative, tooltip. This means that you can move your mouse over any value (or scatter plot) and get any number of different information back out! Nifty.

```
def filt_hexbins(hexbin_stats, min_threshold=0.0):

    from copy import deepcopy

    filt_hexbin_stats = deepcopy(hexbin_stats)
    temp_len = len(filt_hexbin_stats['freq_by_hex'])
    filt_array = [i > min_threshold for i in filt_hexbin_stats['freq_by_hex']]
    for k, v in filt_hexbin_stats.items():
        if type(v) != int:
            if len(v) == temp_len:
                filt_hexbin_stats[k] = [v[i] for i in range(temp_len) if filt_array[i]]

    return filt_hexbin_stats


filt_league_hexbin_stats = filt_hexbins(league_hexbin_stats)

xlocs = filt_league_hexbin_stats['xlocs']
ylocs = filt_league_hexbin_stats['ylocs']
accs_by_hex = filt_league_hexbin_stats['accs_by_hex']
freq_by_hex = np.array([min(max_freq, i) for i in filt_league_hexbin_stats['freq_by_hex'

hexbin_text = [
        '<i>Accuracy: </i>' + str(round(accs_by_hex[i]*100, 1)) + '%<BR>'
        '<i>Frequency: </i>' + str(round(freq_by_hex[i]*100, 2)) + '%'
        for i in range(len(freq_by_hex))
]

fig = go.Figure()
```

```
draw_plotly_court(fig)
fig.add_trace(go.Scatter(
    x=xlocs, y=ylocs, mode='markers', name='markers',
    marker=dict(
        size=freq_by_hex, sizemode='area', sizeref=2. * max(freq_by_hex) / (11. ** 2), s
        color=accs_by_hex, colorscale=colorscale,
        colorbar=dict(
            thickness=15,
            x=0.84,
            y=0.87,
            yanchor='middle',
            len=0.2,
            title=dict(
                text="<B>Accuracy</B>",
                font=dict(
                    size=11,
                    color='#4d4d4d'
                ),
            ),
            tickvals=[marker_cmin, (marker_cmin + marker_cmax) / 2, marker_cmax],
            ticktext=ticktexts,
            tickfont=dict(
                size=11,
                color='#4d4d4d'
            )
        ),
        cmin=marker_cmin, cmax=marker_cmax,
        line=dict(width=1, color='#333333'), symbol='hexagon',
    ),
    text=hexbin_text,
    hoverinfo='text'
))
fig.show(config=dict(displayModeBar=False))
```

basketball_plots_hexbin_plot_4 hosted with ❤ by GitHub view raw





Comparing datasets + adding bitmaps to the figure

For completeness, let's look at one last example where we will compare data of a team against the league average data. This example will also allow us to look at diverging colour scales as well, which are very handy.

We'll also add a bitmap of a team logo to the figure to complete the look.

Load the data, just as we have done previously, but this time loading the file for the Hoston Rockets.

```
teamname = 'HOU'
with open('srcdata/hou_hexbin_stats.pickle', 'rb') as f:
    team_hexbin_stats = pickle.load(f)
```

For this plot, we're going to keep the original team data for everything except the shot accuracy. We will convert the shot accuracy data to be relative, meaning we'll compare it against the league average.

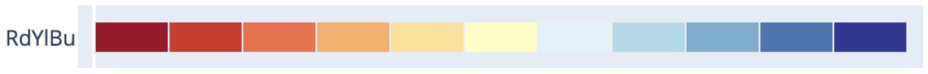
To do this, we use:

```
from copy import deepcopy
rel_hexbin_stats = deepcopy(team_hexbin_stats)
base_hexbin_stats = deepcopy(league_hexbin_stats)
rel_hexbin_stats['accs_by_hex'] = rel_hexbin_stats['accs_by_hex'] -
base_hexbin_stats['accs_by_hex']
```

You might be wondering why I'm using `deepcopy`. The reason is that due to the way python works, if I had simply copied `team_hexbin_stats` to `rel_hexbin_stats`, any modifications to the values in `rel_hexbin_stats` would have also applied to the original dictionary also (read more here).

I could have just modified `team_hexbin_stats`, since we will not be using it again in this tutorial, but it's just bad practice and can lead to confusion.

Using the modified `rel_hexbin_stats`, we can go on with rest of the processing — you know the drill by now. The only changes are to the colourmap, to a diverging map of `RdYlBu_r` (reversed as I want red to be 'good), and change the scale to be from -10% to 10% as we show relative percentages.



RdYlBu colourmap

Lastly, let's add the team logo. Basketball-reference has a handy, standardised URI structure for the team logos, so I simply refer to those with the 'teamname' variable. And then, it's a simple matter of adding the images you desire with the `.add_layout_image` method, specifying its size, placement and layer order.

The entire code for this version looks like this:

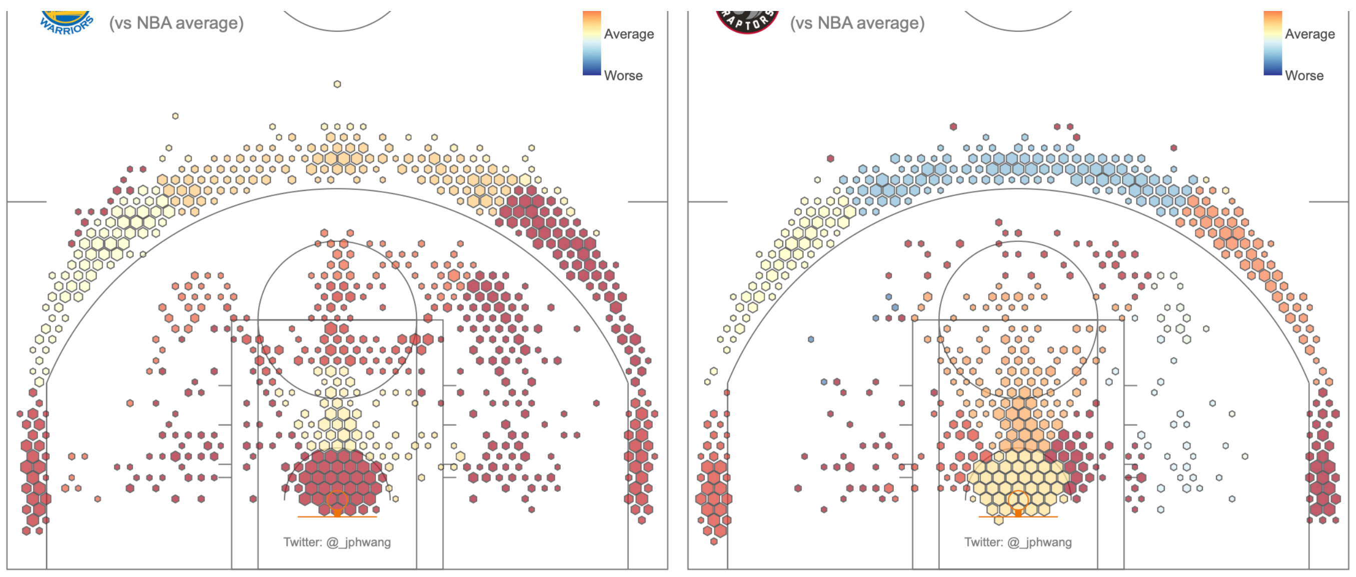
```
from copy import deepcopy
teamname = 'TOR'
with open('basketball_plots/srcdata/' + teamname.lower() + '_hexbin_stats.pickle', 'rb')
    team_hexbin_stats = pickle.load(f)

rel_hexbin_stats = deepcopy(team_hexbin_stats)
base_hexbin_stats = deepcopy(league_hexbin_stats)
rel_hexbin_stats['accs_by_hex'] = rel_hexbin_stats['accs_by_hex'] - base_hexbin_stats['a
rel_hexbin_stats = filt_hexbins(rel_hexbin_stats, min_threshold=0.0)

xlocs = rel_hexbin_stats['xlocs']
ylocs = rel_hexbin_stats['ylocs']
accs_by_hex = rel_hexbin_stats['accs_by_hex']
freq_by_hex = np.array([min(max_freq, i) for i in rel_hexbin_stats['freq_by_hex']])

colorscale = 'RdYlBu_r'
```

```
marker_cmin = -0.05
marker_cmax = 0.05
title_txt = teamname + ":<BR>Shot chart, '18-'19<BR>(vs NBA average)"

hexbin_text = [
        '<i>Accuracy: </i>' + str(round(accs_by_hex[i]*100, 1)) + '% (vs league avg)<BR>
        '<i>Frequency: </i>' + str(round(freq_by_hex[i]*100, 2)) + '%'
        for i in range(len(freq_by_hex))
]
ticktexts = ["Worse", "Average", "Better"]
logo_url = "https://d2p3bygnnzw9w3.cloudfront.net/req/202001161/tlogo/bbr/" + teamname +

fig = go.Figure()
draw_plotly_court(fig, fig_width=600)
fig.add_trace(go.Scatter(
    x=xlocs, y=ylocs, mode='markers', name='markers',
    text=hexbin_text,
    marker=dict(
        size=freq_by_hex, sizemode='area', sizeref=2. * max(freq_by_hex) / (11. ** 2), s
        color=accs_by_hex, colorscale=colorscale,
        colorbar=dict(
            thickness=15,
            x=0.84,
            y=0.87,
            yanchor='middle',
            len=0.2,
            title=dict(
                text="<B>Accuracy</B>",
                font=dict(
                    size=11,
                    color='#4d4d4d'
                ),
            ),
            tickvals=[marker_cmin, (marker_cmin + marker_cmax) / 2, marker_cmax],
            ticktext=ticktexts,
            tickfont=dict(
                size=11,
                color='#4d4d4d'
            )
        ),
        cmin=marker_cmin, cmax=marker_cmax,
        line=dict(width=1, color='#333333'), symbol='hexagon',
    ),
    hoverinfo='text'
))

fig.update_layout(
    title=dict(
        text=title_txt,
        y=0.9,
        x=0.19,
        xanchor='left',
        yanchor='middle'),
    font=dict(
        family="Arial, Tahoma, Helvetica",
        size=10,
        color="#7f7f7f"
    ),
    annotations=[
        go.layout.Annotation(
            x=0.5,
            y=0.05,
            showarrow=False,
            text="Twitter: @_jphwang",
            xref="paper",
            yref="paper"
        ),
    ],
)

fig.add_layout_image(
    go.layout.Image(
        source=logo_url,
        xref="x", yref="y", x=-230, y=405, sizex=50, sizey=50,
        xanchor="left", yanchor="top",
        sizing="stretch", opacity=1, layer="above"))

fig.show(config=dict(displayModeBar=False))
```

basketball_plots_hexbin_plot_5.txt hosted with ❤ by GitHub view raw

And look at these shot charts — glorious! I can't embed an interactive version here, but I'm pretty happy with how they turned out.

> *(Edit: here's a live demo (for TOR) — also for GSW, HOU, SAS and NYK)*



Shot charts for the 2 teams in last year's NBA Finals.

As you can see, it's really quite straightforward to plot spatial data using Plotly. What's more, between Plotly Express and Plotly, you have a range of options that are sure to meet your requirements, whether they be quick & dirty data exploration, or in-depth, customised visualisations for your client.

Try it out for yourself, I've plotted the Finals' teams, but you can have a look at your own teams in the repo. I include data for all the teams in it. I think you'll be impressed by how easy Plotly makes visualisation, while being incredibly powerful and flexible.

. . .

That's all from me — I hope you found it useful, and hit me up if you have any questions or comments!

> *I'm not sure what the best way to structure python files for a tutorial like this is, but I've just written everything in one file, so you can just comment / uncomment as you'd like and follow along.*

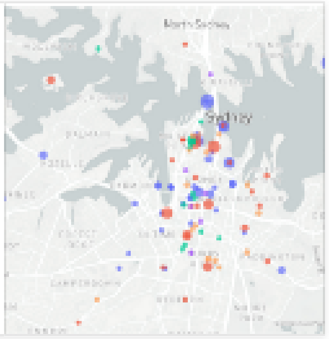
. . .

If you liked this, say 👋 / follow on twitter, or follow for updates. I also wrote last week about producing interactive maps with Plotly.

Interactive maps with Python, pandas and Plotly: following bloggers through Sydney

In this article and another few, I will explore Python and Plotly to put together a few different awesome looking…

towardsdatascience.com



Programming Sports Python Data Science Coding

Medium

About Help Legal