Blazing the Nets

Saiem Gilani

March 31, 2021

1 Introduction

The objective was to create interactive NBA visuals in JavaScript with data being pulled from the back-end to the front-end of the application. While I have 20 years of ESPN data already pulled and stored in a public repository, I thought I should familiarize myself with the NBA Stats API endpoints and the player tracking data available. After doing some research, I found that the league is blacklisting cloud provider IP addresses from accessing the API and while I have a work-around mapped out, I decided it best to avoid implementing that for now. I chose to go after re-creating *Buckets*, Peter Beshai's visualization from 2014 using D3.js and Angular. I was really just trying to accomplish the same type of analysis using D3v6 and React.

2 Tech Talk

The stack I chose in a pinch roughly followed as so:

Task	**Tool**
Extract, Transform, Load	Node.js
Back-End	Firebase Storage
Front-End	React
State Management	React-Hooks
Front-End Visuals	D3.js
UI Components	Material-UI
Deployment	GitHub Actions
Hosting	Firebase

2.1 Extract, Transform, Load

In the end, I had no issues getting access to the data via Node.js as in `etl/`, which scrapes roughly 14 endpoints for the most recent 6 seasons of data. Due to the time constraints, I

primarily wound up working with the `shotchartdetail` endpoint, which allowed me to get the raw data to calculate shot bins and compare player averages to the league average from that distance. I spent a reasonable amount of time just searching and scraping of the endpoints to find the endpoints which held the level of detailed data I was interested in to make this visualization work.

2.2 Back-end

Used Firebase Storage, created the collections necessary to pull the data, I would have really loved to do more here to actually organize the data in formats that lent themselves to easier drill-downs, but time was of the essence. I wanted to understand the options when it comes to React's state management and interactivity. So I used React Hooks paired with Axios to manage most of the interactions with the storage data. It could have just as easily been managed through Express.js and just building the routes that way. While I learned alot about React Hooks from this project, I really wish I had used Express.js.

2.3 Front-End & Visualization

Personally, I generally have little problem developing API's or connecting back-end analysis and models to front-end visualizations, so I tried to gain something from this by making a really interactive visual that works in React. I managed to connect the visuals via voronoi accessor variables and sending dispatches on hover movements to the other visuals so that the highlighting stays in sync as below.

Figure 2.1: Blazing-the-Nets

3 Linked Visuals

So I think a rough outline of the functionality of *Buckets* was achieved, allowing us to highlight 5 visuals according to the dimension they share in common, distance from the basket. Moreover, we also see that they visuals are all also linked chromatically (perhaps redundantly) on the basis of % difference from League Average.

3.1 Shot Chart

The first component is your standard hexbin shotchart:

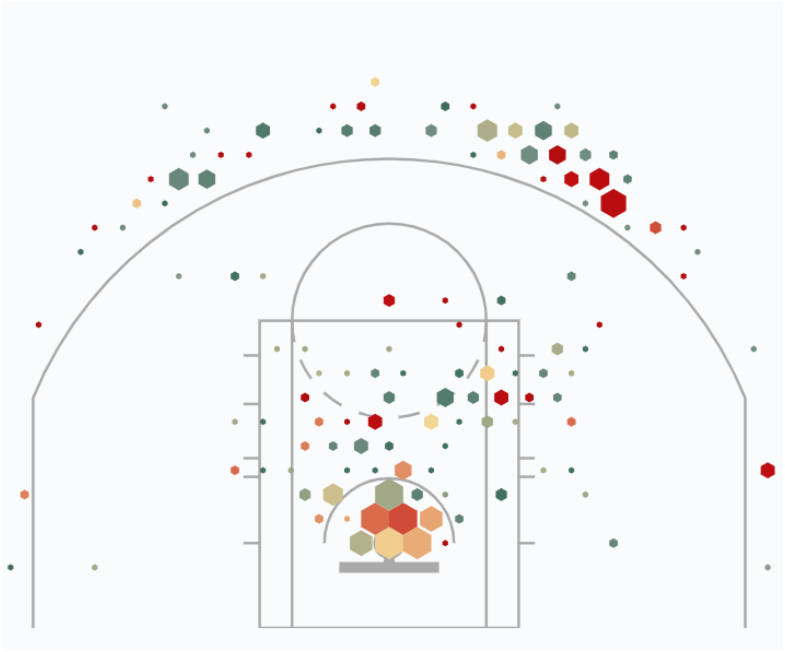

Figure 3.1: Blazing-the-Nets: Shot Chart

3.2 SHOOTING SIGNATURE

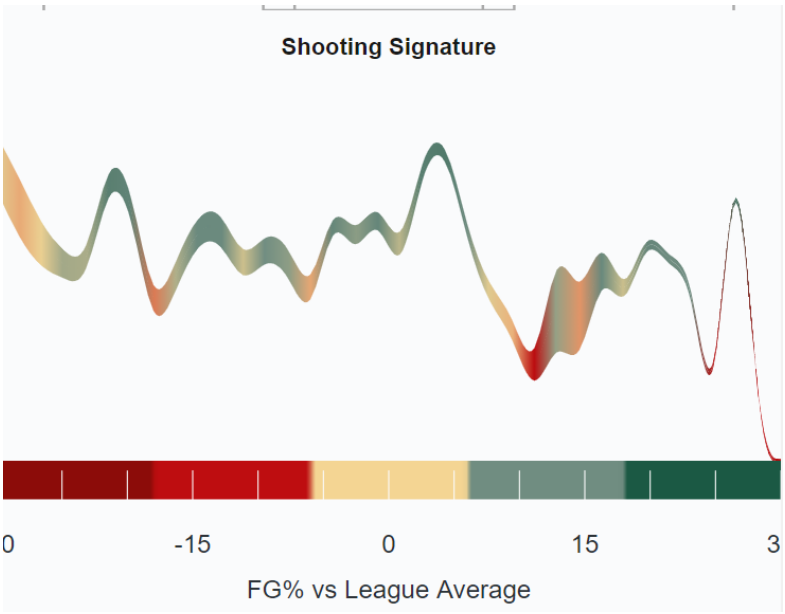

Figure 3.2: Blazing the Nets: Shooting Signature

3.3 Shot Frequency and Efficiency by Distance

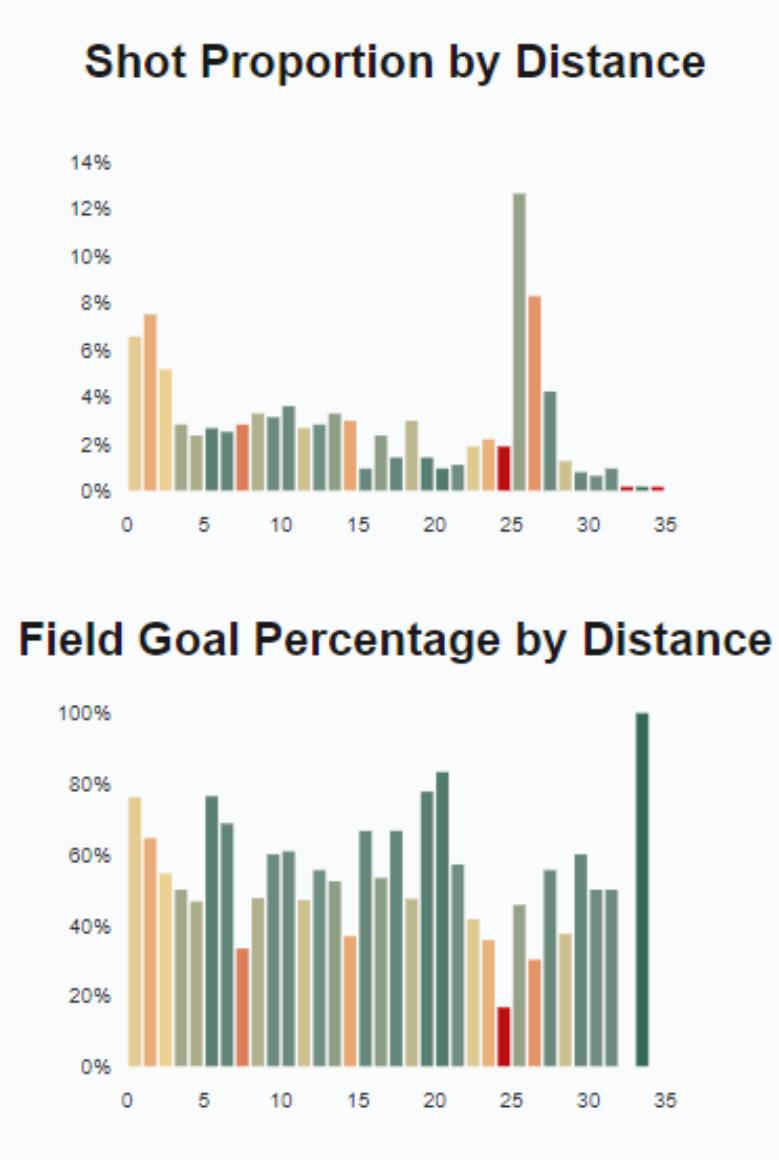

Figure 3.3: Blazing the Nets: Shot Frequency and Efficiency by Distance

3.4 Shot Frequency and Efficiency by Side

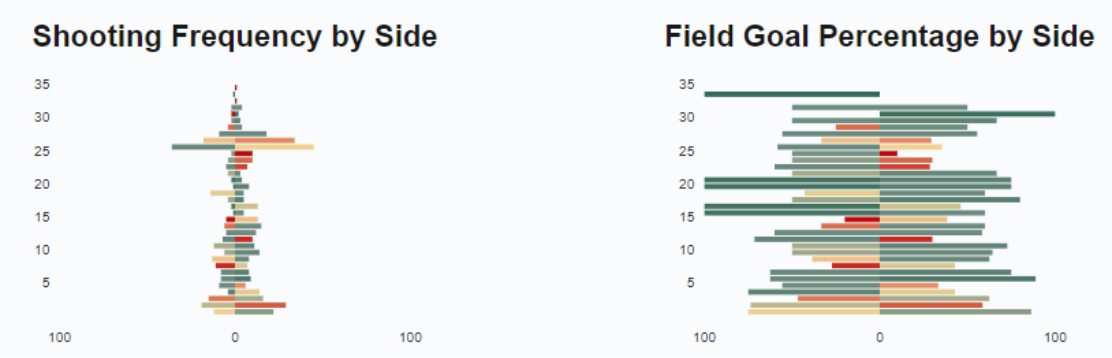

Figure 3.4: Blazing the Nets: Shot Frequency and Efficiency by Side

4 Conclusion

While there is still a great deal of work to be done to get this visual into a workable cohesive application, the framework has been set up to create many kinds of linked visuals and layerings for charts. The techniques I learned from attempting to replicate it have been a great foundation builder on the possibilities of creating high quality custom interactions and I look forward to working with more React visualization libraries.