Webapp for Automated Visualization: Matplotlib Plotting for NBA Shooting Stats, Heatmaps, and More

Emre Bayazitoglu

Albert Ludwig University of Freiburg, Freiburg, Germany.

emre.bayazitoglu@students.uni-freiburg.de;

Abstract

stats.nba.com provides a wealth of statistics each season, including data on every player's shooting positions per game. To enhance the visualization of these shooting positions' x- and y-coordinates, I developed a shooting position map using Matplotlib, which plots markings on a pre-drawn basketball court. Motivated by my passion for mathematics and extensive experience with Matplotlib, I also incorporated various mathematical and illustrative plots into this project, such as trigonometric functions, simple functions dependent on one variable, bar charts, and pie charts.

This report offers a review of my experiences, outcomes, and the technologies utilized in my final project for the "Missing Semester" course. Additionally, it provides insights into code examples and methodologies employed in the development and sharing of the final software.

For access to the code and detailed setup instructions, please visit the GitHub repository of this project.

Keywords: Vue.js, Python, Flask, Docker, Matplotlib, automated plotting

1 Introduction

The game of basketball is multifaceted, making it crucial to visualize data for analyzing individual players or teams each season. Over the past 20 years, the NBA has collected detailed data to facilitate analysis, with datasets stored in CSV files. To enable data communication, I developed a Python script to transfer this data into a database managed by PySQLite. This database serves as the foundation for a faster and more efficient data retrieval process. Using this database, users can select specific players for each season via a Vue.js frontend. Following user input, they can choose to display *made shots* or *missed shots* as options, which are then analyzed using a Matplotlib Python script. This generates plots that users can utilize for basketball player analysis. All in all, a variety of technologies are employed to automate the process of generating meaningful and customizable plots.

For users that are not interested in basketball, the web app also offers options to plot mathematical functions and charts. These features are accessible through the frontend, where users fill out a form. Upon form submission, the automation starts, making-use of useful Python libraries and frameworks like Flask, a web framework used for communication between the frontend and backend, and NumPy for evaluating mathematical functions inputted by the user.

This report is structured to guide users through the Matplotlib scripts, starting from the frontend via the interface between the frontend and backend. Finally, the process for dockerizing the software is explained.

2 Methodology

The project requirements were two fusion two topics from the "Missing Semester" lecture, and I chose the ones that interested me the most: plotting with Matplotlib and automation. Additionally, to facilitate project sharing, I used Docker with Dockerfiles and a *docker-compose.yml* configuration.

The idea came when I discovered the vast dataset provided by stats.nba.com, so I decided to create a tool that plots shooting maps for NBA players across different seasons. To achieve this, I began by designing a user-friendly webpage using Vue.js and PrimeVue for the frontend and Flask for handling requests between the frontend and backend.

Having established the frontend-backend interface, I proceeded to develop the backend infrastructure. This included developing a Python script to initialize a database from a CSV file and a Matplotlib script for generating the shooting maps. Recognizing the central role of the database in NBA plotting, priority was set to its creation at the beginning of the dockerization.

Following the frontend and backend design, I focused on ensuring all dependencies were met. To ensure this, I created Dockerfiles for both the frontend and backend, composing them through a *docker-compose.yml* configuration. This approach simplifies dependency management and facilitates project deployment for users.

3 Automation: Vue.js Frontend and Python Backend

Several technologies work together to automate the process of data management, processing, and visualization in the project:

- **Graphical User Interface (GUI):** The GUI runs on a local host and provides a user-friendly interface for interaction. It allows users to input data and visualize results conveniently.

- **Flask Web Server:** Flask is a lightweight web framework for Python [1], used in this project to handle HTTP requests between the frontend and backend.
- **PySQLite-managed Database:** PySQLite is a Python library that provides an interface for interacting with SQLite databases. In this project, it manages the database operations, such as data storage, retrieval, and manipulation. [2]

3.1 Vue.js Frontend

Once the project is set up successfully, the user can refer to the GUI to interact with the software. The software is structured with a panel menu in which the user can choose between several options:

- NBA shooting map
- simple plotting
- bar charts
- pie charts
- trigonometric functions.

The design of the webapp is intended to be intuitive and easy to use. The user simply needs to fill out the forms and is provided with the corresponding plots. In general, communication is implemented using axios. Once a form is filled out, a POST method is executed to provide the backend with the user inputs and receive, most commonly, a plot as a response. The communication is being maintained by an proxy, which was implemented like this:

```
const { defineConfig } = require("@vue/cli-service");
module.exports = defineConfig({
  transpileDependencies: true,
  devServer: {
    proxy: {
      "/api": {
        target: "http://172.18.0.2:5000",
        pathRewrite: { "^/api": "" },
      },
    },
  },
});
```

Since the GUI and the backend web server are not running on the same host (they run in different container), I needed to proxy API requests to the API server. [3]

- `const { defineConfig } = require("@vue/cli-service");`: This line imports the `defineConfig` function from the `@vue/cli-service` package.
- `transpileDependencies: true`: This configuration option tells Vue CLI to transpile dependencies using Babel [3]. When set to `true`, all dependencies in the `node_modules` directory will be transpiled to ensure compatibility with older browsers [3].

- `devServer`: This section configures the development server options. Which are:
  - `proxy`: Used to forward requests from the used axios methods in the web app to the matching Flask routes.
  - `"/api"`: This line specifies the prefix of the requests that should be proxied eg.:

    ```
    axios.post("/api/postBar", data) \\ send data to the speciefied
                                     \\ prefix "/api/postBar"
    .then((response) => {
        console.log(respone) \\ log the response
    }).catch((error) => {
        console.log(error);  \\ log possible errors
    })
    ```

  - `target: "http://172.18.0.2:5000"`: This line specifies the target URL to which requests matching the specified prefix should be forwarded. In this case, requests starting with `/api` will be forwarded to `http://172.18.0.2:5000`. Which is the static IP of the docker container.
  - `pathRewrite: { "^/api": "" }`: Finally, the `/api` prefix is removed before sending the request to the Flask-route.

Overall, this configuration file sets up a development server for a Vue.js project and configures it to proxy requests to a backend server running at http://172.18.0.2:5000 which is the static ip of the docker container in which the Flask backend server runs.

3.1.1 NBA Shooting Map

The NBA shooting map option features a drop-down component allowing users to select an NBA season ranging from 2004 to 2023. Due to faulty data within the datasets, seasons 2020, 2021, and 2022 are excluded. Additionally, the interface offers two sets of radio buttons. The first set enables users to toggle between viewing options, including a *heat map* or a *position marking map*. The second set provides users with the choice between displaying a plot for *made shots* or *missed shots*.

To enhance usability and organization, the drop-down component for player selection is structured to first prompt users to choose a team before presenting a list of available players. This was realized using the cascade-select component of PrimeVue's component library.

3.1.2 Simple Plotting

The simple plotting option contains of six input fields. The first input field requires users to input a function in a notation interpretable by Python. For example, x^2 should be entered as *x**2*. This notation ensures compatibility with Python's syntax for function evaluation. Following the function input, users specify the x-range of the plot using the start and stop input fields. These parameters define the range over which the function will be plotted. Users may use the remaining three input fields to label the x- and y-axes and to assign a title to the plot.

3.1.3 Bar Charts and Pie Charts

The bar chart and pie chart options feature input fields and a table for enhanced customization and data presentation. For the bar chart option, users encounter three input fields for axis and plot labeling. These fields enable users to specify labels for the x-axis, y-axis, and the plot itself, ensuring clear and informative visualizations. Furthermore, the pie chart option includes a single input field for labeling the plot, as pie charts do not have axes like bar charts.

The table structure for both options is organized as depicted in Table 1:

Column	**Functionality**	**Changable in Edit Mode?**
ID	displays the row number.	No.
Name	displays the label of the bar or slice.	Yes.
Value	displays the ”height” of the bar chart/portion of the slice of the pie chart (all values must sum up to 100 for the pie chart).	Yes.
Color	the user can select a color for the corresponding bar or slice.	Yes. A color picker appears.

Table 1: Functionality of table columns for bar/pie chart

Every row contains a button with a stylus as a label. Once the user clicks on that stylus, the table changes into edit mode, which means that the user can change the row elements.

3.1.4 Trigonometric Funtions

In the trigonometric plotting option, users can plot trigonometric functions with customizable parameters. This option presents seven input fields and one set of radio buttons to define the mode, supporting sine, cosine, and tangent functions. Upon selecting this option, users are prompted to specify a title for the plot. Similar to the simple plotting option, users define the start and stop values for the plot range. The remaining input fields are labeled as a, b, c, and d. While these labels may seem imprecise, they correspond to parameters in the following equation:

$$f(x) = a \times \sin(b \times x + c) + d$$

- a denotes the amplitude, controlling the vertical stretching or compression of the function [5].
- b denotes the period, representing the horizontal length of one cycle of the function [5].
- c denotes the phase shift, determining the horizontal displacement of the function [5].
- d denotes the vertical shift, indicating the vertical displacement of the function [5].

These parameters allow users to customize the shape and behavior of the plotted trigonometric function according to their preferences.

3.2 Python Backend

The backend is divided into three individual but yet dependent systems.

- Flask app to handle incoming requests from the backend
- PySQLite script to create and manage databases
- Matplotlib script to generate the plots.

The Flask and PySQLite scripts are covered in this section, while the Matplotlib scripts are covered in their own sections.

3.2.1 Flask App

The Flask application provides routes to handle various plotting functionalities. For NBA shooting statistics visualization, I created a dedicated route *postPlotNBA* to process user requests, extract relevant data based on user inputs (such as selected player, season, and shot type - *missed shots* or *made shots*), and generate corresponding plots using Matplotlib. Similar routes were implemented for trigonometric function plots *postTrig*, simple function plots *postSimplePlot*, bar charts *postBar*, and pie charts *postPie*. The Flask application script calls the instances of the plotting classes, which are covered in the Matplotlib section. The workflow here is as follows:

1. Receive the HTTP-request from the frontend.
2. Pass the data to the instances of the plotter classes.
3. Receive the plots from the instances.
4. Respond to the HTTP-requests with the plots

3.2.2 PySQLite

The script that initially sets up the local database uses PySQLite. It is a script that has a simple purpose: create and fill a datatable. The workflow is as follows:

1. create an empty database
2. connect to that database
3. create a datatable *nbashotsperplayer*, using an SQL query

   ```
   "CREATE TABLE nbashotsperplayer (SEASON_1 int,
   TEAM_ID int, TEAM_NAME text, PLAYER_ID int,
   PLAYER_NAME text, EVENT_TYPE text, SHOT_MADE text,
   LOC_X double, LOC_Y double);"
   ```

4. open and read the CSV file, extract the columns needed, and insert them into the datatable *nbashotsperplayer*.

```
"INSERT INTO nbashotsperplayer (SEASON_1, TEAM_ID,
TEAM_NAME, PLAYER_ID, PLAYER_NAME, EVENT_TYPE, SHOT_MADE,
LOC_X, LOC_Y) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);"
```

4 Matplotlib

The Matplotlib section of the project consists of various classes, each dedicated to a specific type of plot. Instances of these classes are called within the respective routes of the Flask application to handle plot generation based on user requests. The functionality of each class is elaborated upon in the corresponding subsections.

4.1 Simple Plots Depending on One x

This *SimplePlot* class is designed to create and save simple plots based on user-defined equations that depend on one variable: x. The constructor initializes the instance with parameters representing the equation, start and stop values for the x-axis, x and y-axis labels, and the plot title. A method within this class generates the plot based on the provided equation and saves it as a PNG image. First, the method generates an array of evenly spaced numbers (for x) between the start and stop values. When evaluating the provided equation string, NumPy's eval function comes in handy to calculate the corresponding y-values. The calculated points from the function are then projected onto the coordinate system. Finally, a little styling is done to improve the appearance.

4.2 Trigonometric Functions

Using the provided parameters - amplitude a, frequency b, phase shift c, and vertical shift d - the method constructs the function and plots it over a specified range of x-values. The resulting plot is saved as a PNG image with the specified title. NumPy provides all three trigonometric functions as methods in the library. All I needed to do was provide the user-specified values. So the implementation of this was rather easy.

4.3 Pie Charts and Bar Charts

This class uses the provided parameters, such as the names of each slice with corresponding values and customizable colors. Upon instantiation, the class constructs the pie chart, ensuring each slice represents its proportion of the whole. Afterwards, to enhance readability, annotations to display percentage values within each segment were included. The resulting plot is equipped with the user-inputted title and legend.

4.4 NBA Shooting

The biggest effort in plotting was put into the NBA shooting maps. The NBA shooting heat and marking maps use a Matplotlib-drawn basketball half-court and map the x- and y-coordinates of the selected players shot on the half-court. The half-court borders are represented by a rectangle. The three-point line is a composition of two straight lines combined with a semicircle. In this three-point box, there is the hoop, which is represented by a circle and a straight line; a circle, which outlines the free-throw line;

and finally, the restricted zone, which is also a rectangle. The x-values for the half-court range from -250 to 250, where the origin is the center of the hoop. The y values range from -47.5 to 422.5. Since the x- and y-coordinates of the dataset only range for x from -30 to 30 and for y from 0 to 40, I had to scale the coordinates such that they fit into the half-court properly. Depending on whether the heatmap represents *made shots* or *missed shots*, the title of the plot is dynamically adjusted to reflect the corresponding data.

4.4.1 Marking Map

The marking map uses red crosses if the *missed hots* option is selected in the frontend and green circles if the *made shots* option is selected. After the scaling happens, the points are mapped on the basketball court.

4.4.2 Heat Map

I have created a heatmap plot using the Seaborn library, which depicts the distribution of the data points on a basketball court. The heatmap is customized with a specific colormap style ("YlOrRd_r") to highlight variations in density. Additionally, the x and y limits are set to ensure the entire half-court is visible.

5 Docker

In order to setup the project locally, the user has to have Docker and Docker Compose installed. In this project, Docker is used to make the setup as easy as possible for the user. The project setup consists of including the CSV file with the data of the shooting positions into a project folder and then running the command:

```
docker-compose up
```

in the root folder of the repository.

In the Git repository of this project, there are two Dockerfiles, which are composed using a *docker-compose.yml* configuration. The Dockerfiles set up the environment for the frontend and the backend. For the backend, all the dependencies are installed using *pip*, which is the package installer for Python. In the Dockerfile, the dependencies are installed into the environment like this:

```
RUN python3 -m pip install <package>
```

The image that is used for the environment is the Python 3.11 image. After installing the dependencies, the backend folder, which contains all the Python scripts, is copied into the Docker container and then executed. The approach for the frontend is similar. On the frontend, the dependencies are installed using *npm*, which is the package installer specialized for Javascript environments. All the dependencies are specified in the *package.json* and are simply installed using:

```
RUN npm install
```

Yet again, the folder containing the frontend code is copied into the container. The frontend is then run and served on local host with port 8080 using the command:

```
CMD["npm", "run", "serve"]
```

The *docker-compose.yml* configuration is then executing the frontend and backend services. Here, the ports of the container are mapped to the ports of the host machine for both services.

6 Results

After successfully running:

```
docker-compose up
```

in the root directory of the project folder, we can get access to the frontend by opening http://localhost:8080 in the browser. This is guaranteed to work for a browser in the browserlist. The figures 1-6 depict screenshots of the web app:

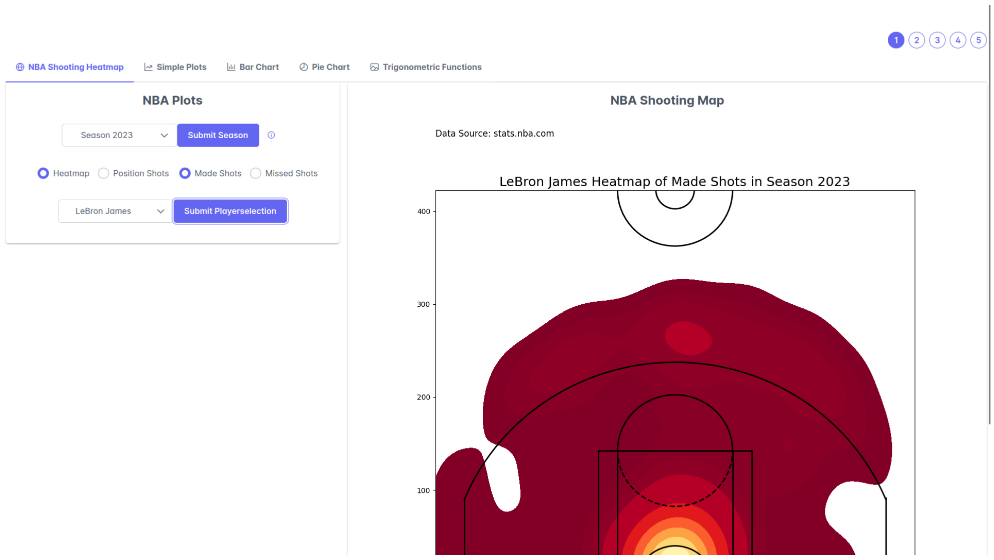


Fig. 1: Option 1 view: NBA Shooting Heatmap

7 Consclution

In this project, I learned a variety of skills and gained valuable experience in several areas:

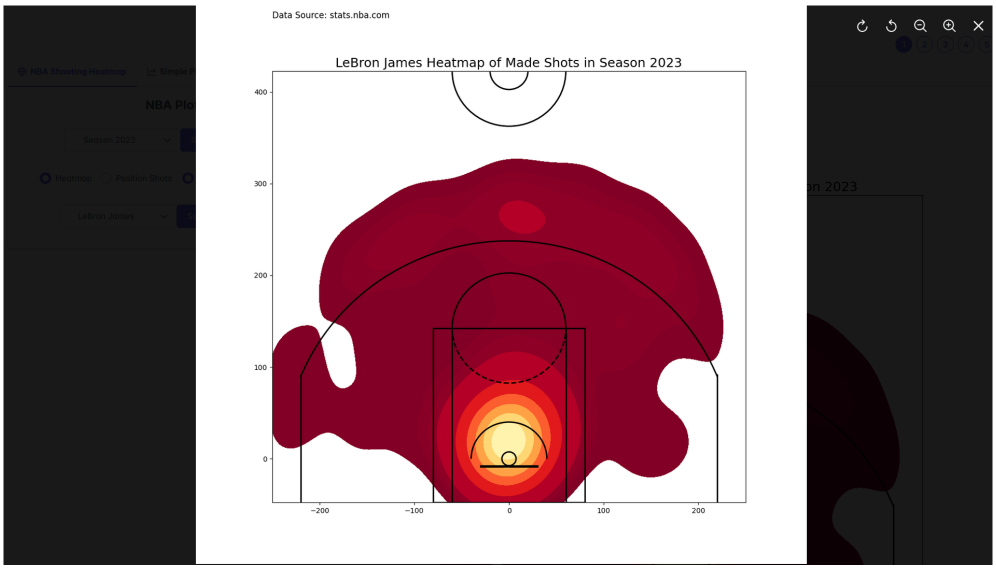
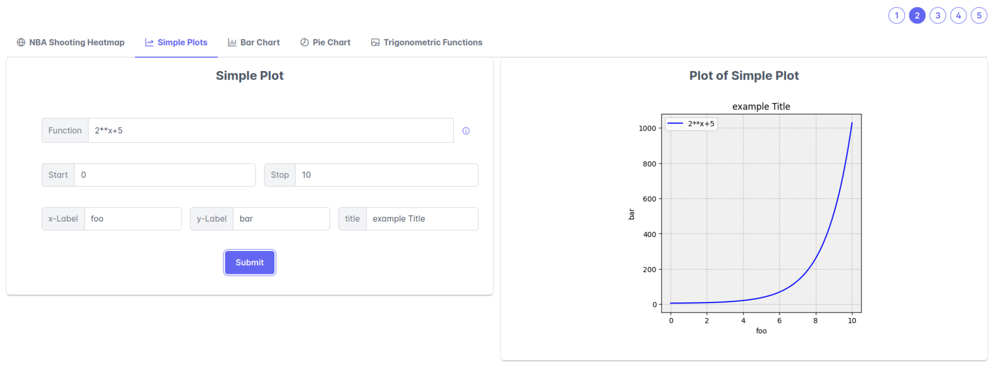

Data Source: stats.nba.com
LeBron James Heatmap of Made Shots in Season 2023
400
300
200
100
0
−200
−100
0
100
200

Fig. 2: Images on webapp are clickable - zoomed view



Simple Plot

Function: 2**x+5

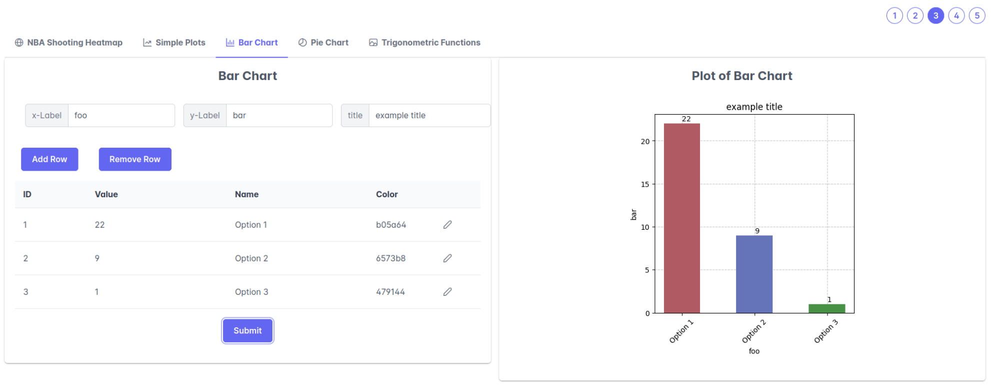
Start: 0 Stop: 10

x-Label: foo y-Label: bar title: example Title

Plot of Simple Plot

Fig. 3: Option 2 view: Simple Function

- **Frontend Development**: I worked with Vue.js and PrimeVue to create a user-friendly interface for interacting with a Matplotlib script.
- **Backend Development**: I used Flask to build the backend of my application. This involved creating routes, handling HTTP requests, and interacting with a PySQLite database to store and retrieve data.
- **Data Visualization**: I utilized Matplotlib, Seaborn, and other Python libraries to generate various types of plots and visualizations, such as bar charts, pie charts,

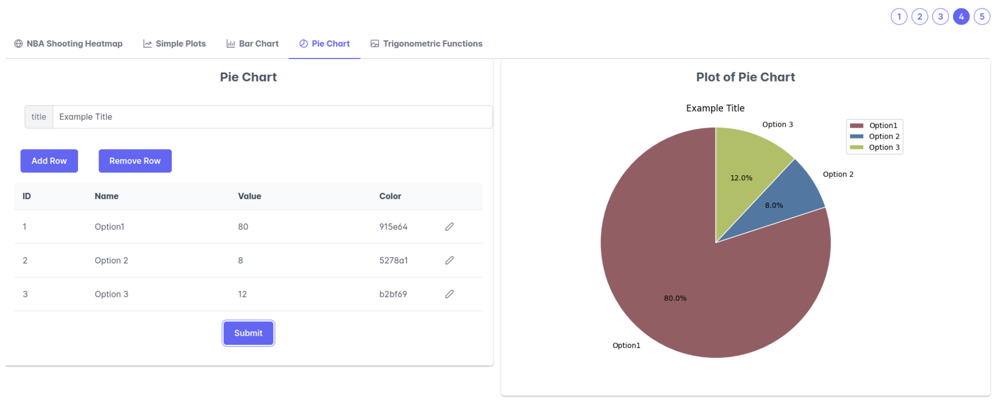


Bar Chart

x-Label: foo   y-Label: bar   title: example title

ID	Value	Name	Color
1	22	Option 1	b05a64
2	9	Option 2	6573b8
3	1	Option 3	479144

Fig. 4: Option 3 view: Bar Charts

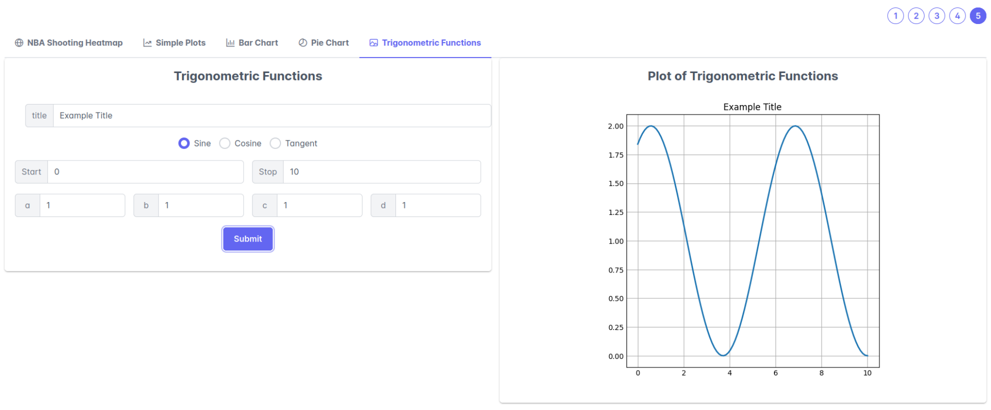


Pie Chart

title: Example Title

ID	Name	Value	Color
1	Option1	80	915e64
2	Option 2	8	5278a1
3	Option 3	12	b2bf69

Fig. 5: Option 4 view: Pie Charts



Trigonometric Functions

title: Example Title

[✓ Sine] Cosine Tangent

Start: 0   Stop: 10

a: 1   b: 1   c: 1   d: 1

Fig. 6: Option 5 view: Trigonometric Function

Topics	
Liinux	Manjaro Linux is used as an operating system to test and develop the software. Other than that, there is nothing special to report.
Emacs	Not used.
VSCode	Used as a source-code editor with some extensions to improve code quality.
Nvim	Nothing special to report.
Git	Used a Git repository to version and store code. Also used to submit the project-assignment.
Automation	Combined multiple technologies to automate the plotting process.
Docker	Wrote Dockerfiles and used the concept of Docker Compose for the project-assignment.
Matplotlib	Used for complex plotting.
Gnuplot	Not used.
LaTex	Used to report on the project.
LLM's	Not used.

heatmaps, and trigonometric plots. This involved processing data and presenting it in a visually appealing and informative way.

- **Dockerization**: I dockerized my application, which involved creating a Dockerfile to specify the environment and dependencies needed to run the application and using *docker-compose.yml* configuration to manage two containerized services.
- **Project Management**: Throughout the project, I gained experience in project planning, task prioritization, and time management. In the review part, I also learned to analyze and comment on other developers codes.

Overall, this project provided me with a learning experience in web development, data visualization, database management, and containerization, equipping me with valuable skills that can be applied to future projects and in my career as a software developer.

8 References

[1] Flask Quickstart - Writting A Minimal Application with Flask

[2] sqlite3 — DB-API 2.0 interface for SQLite databases - A Tutorial for getting started with sqlite3.

[3] vue CLI Configuration Reference - Guide on Global CLI Config

[4] seaborn.kdeplot - Plot univariate or bivariate distributions using kernel density estimation.

[5] Math is Fun - Amplitude, Period, Phase Shift and Frequency

Acknowledgements. A special thanks goes out to the "Missing Semester" team, who did a very good job organizing such a practical course, where I was allowed to learn concepts that are very much used in the industry. During this course and the final project, I enhanced my knowledge a lot with interesting slides and exercises.