Introduction to Data Visualization

Connecticut Sports Analystics Symposium 2025

Rahul Manna

April 11, 2025

Table of contents

1 Introduction

Visualizing data, particularly in sports analytics, provides valuable insights for informed decision-making, reveals underlying patterns in the data, and enhances communication among stakeholders. Leveraging Python's versatility and powerful data visualization libraries enables the creation of well-crafted visual narratives across diverse domains. Matplotlib, Python's most popular visualization package, offers extensive customization options and precise control, making it a preferred tool for crafting detailed and impactful visualizations. This workshop will introduce Matplotlib's robust plotting capabilities, showcase practical examples of data visualizations in baseball and basketball, and equip participants with versatile techniques applicable across any domain.

1.1 About the Instructor

Rahul Manna is a junior pursuing a dual degree in Statistical Data Science and Mechanical Engineering at the University of Connecticut. He is currently working as a research assistant in the Laboratory for Advanced Manufacturing Reliability (KKim Lab), where he tests materials for implantable bioelectronics and uses Python and Matplotlib to analyze and visualize data. Outside the classroom and workspace, Rahul enjoys Formula One, where the fusion of cutting-edge engineering, advanced statistics, data science, analytics, and human ingenuity drives both the on-track performances and the strategic decisions behind the scenes

1.2 Data

Data used in this workshop is obtained from Baseball Savant and NBA API with the `pybaseball` and `nba_api` packages.

1.3 References

Available at References

2 Data Visualization

2.1 Why Do We Need to Visualize Data?

2.1.1 Humans Process Visuals Faster

- **90% of information transmitted to the brain is visual**
  Our brains are wired to process and interpret visual information much more efficiently than text or numbers alone.

- **We process images 60,000 times faster than text**
  This speed allows us to recognize patterns, trends, and relationships in data almost instantly.

- **Example: A simple chart vs. a raw data table → Which is easier to interpret?**
  Consider a table with hundreds of numbers representing sales over time versus a line chart showing the trend. The chart immediately reveals growth, decline, or seasonality, while the table requires extra effort to interpret.

Source: International Forum of Visual Practitioners (n.d.)

2.1.2 Identifying Trends & Patterns

- **Raw data can be overwhelming, but visuals reveal insights**
  Large datasets are difficult to comprehend in tabular form, but a

well-designed visualization can highlight key trends, relationships, and outliers in seconds.

- **Easier to detect correlations, outliers, and distributions**
  - Correlations: A scatter plot can show how two variables relate to each other.
  - Outliers: Box plots and histograms can highlight anomalies in the data.
  - Distributions: Histograms and density plots reveal the shape of data distributions.

- **Example: A scatter plot of sales vs. marketing spend → Trends appear at a glance**
  A scatter plot can immediately show whether increased marketing spending leads to higher sales, revealing a possible correlation.

2.1.3 Better Decision-Making

- **Helps businesses, researchers, and analysts make data-driven decisions**
  Data visualizations support strategic planning by providing clear evidence to back up decisions.

- **Reduces bias by presenting clear, objective insights**
  A well-constructed visualization minimizes misinterpretation and allows stakeholders to focus on the data rather than personal opinions.

- **Example: Heatmaps in sports analytics → Where does a player score the most?**
  A heatmap of shot attempts in basketball or soccer can reveal a

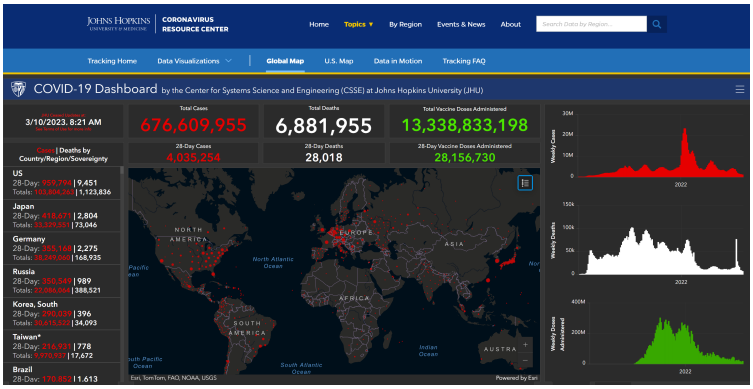
player's preferred scoring areas, helping coaches refine game strategies.

2.1.4 Effective Storytelling

- **Data alone does not engage people, stories do**
  Numbers may contain valuable insights, but without visualization, they can be difficult to communicate to a broad audience.

- **Visualization transforms numbers into compelling narratives**
  Good visualizations create a structured story, guiding the audience through key insights with clear, intuitive graphics.

- **Example: COVID-19 charts → How visualization helped communicate urgency globally**
  Charts from sources like John Hopkins University helped illustrate the rapid spread of COVID-19, emphasizing the need for urgent policy responses.



Source: Johns Hopkins University (n.d.)

2.1.5 The Role of Visualization in Sports

- **Enhances decision-making by optimizing strategies**
  Coaches and analysts use visualized data to determine the best plays, formations, and tactics based on past performance.

- **Identifies patterns and trends in player and team performance**
  Tracking and visualizing performance metrics over time helps identify areas for improvement.

- **Improves fan engagement with accessible insights**
  Visualizing metrics and statistics in an intuitive way allows fans to better understand game dynamics, player performance, and team strategies, making the sport more engaging and interactive.

Source: QuickStart (2024)

2.2 What Makes a Good Plot?

- **Clarity & Simplicity → Avoid clutter, use clear labels** A good plot should be easy to interpret at a glance, with well-labeled axes and an intuitive layout.
- **Accuracy → Represent data truthfully, avoid misleading scales** Avoid using truncated axes or inappropriate chart types that might distort the interpretation of data.
- **Effective Use of Color → Use contrast wisely, avoid red-green combinations, and use colorblind-friendly palettes**
  Many people are colorblind; using color schemes like Viridis or colorblind-friendly palettes ensures accessibility.
- **Right Chart Type → Match the plot to the data**

  - Line charts for trends over time
  - Bar charts for categorical comparisons
  - Scatter plots for relationships and correlations

- **Storytelling → Every plot should highlight key insights** The audience should be able to immediately understand the takeaway message of the visualization.
- **Shapes & Line Styles → Dotted lines for trends, different markers for categories** Customizing markers and line styles makes it easier to distinguish different series in a plot.
- **Resolution & Scalability → High DPI for presentations, vector formats for publications** Ensure visualizations look sharp in different formats, whether on a screen or in print.

3 Python and Visualization

3.1 Python

Python is a high-level, interpreted programming language known for its readability and versatility. It is widely used in data science, machine learning, web development, and scientific computing. Python has a vast ecosystem of packages that extend its functionality. Popular package managers like `pip` and `conda` allow users to install libraries for data analysis, machine learning, web development, and more. For example, `pip install numpy` installs NumPy, a package for numerical computing.

3.2 Installing Python

To install Python, visit the official website python.org and download the latest version. Alternatively, install it using package managers:

- Windows: Install via the Microsoft Store
- macOS: Use Homebrew: `brew install python`
- Linux: Use the system's package manager: `sudo apt install python3`

3.3 Running Python

Once installed, you can run Python in multiple ways:

- **Interactive Mode**: Open a terminal and type `python` or `python3` to start an interactive session.
- **Script Mode**: Write a Python script (e.g., `script.py`) and execute it using `python script.py`.
- **Jupyter Notebook**: Run `jupyter notebook` if Jupyter is installed for an interactive coding environment.
- **VS Code & PyCharm**: Use an IDE like VS Code or PyCharm for development.

3.4 Python Plotting Packages

3.4.1 Matplotlib (Matplotlib Development Team (2024))

- **Versatile and widely used for static, animated, and interactive plots**
  Matplotlib provides extensive tools for creating various types of visualizations, from simple line plots to complex multi-panel figures.

- **Provides full control over plot customization**
  Users can customize every aspect of the plot, including fonts, colors, tick marks, and gridlines.

- **Used for general-purpose plotting in sports analytics**
  Analysts use Matplotlib to visualize player performance, tracking data, and game statistics.

3.4.2 Seaborn (Seaborn Development Team (2024))

- **Built on Matplotlib with enhanced statistical visualization**
  Seaborn simplifies the creation of aesthetically pleasing statistical plots.

- **Ideal for exploring correlations, distributions, and trends**
  Features like `pairplot` and `heatmap` make it easy to identify relationships in data.

- **Useful for analyzing player stats and team performance**
  Seaborn's `boxplot` can compare player performance across different seasons.

3.4.3 Sportypy (SportsDataverse (2024))

- **Designed specifically for sports analytics visualization**
  Provides built-in functions to visualize game-specific elements such as field layouts.

- **Supports court, field, and pitch plotting for multiple sports**
  Makes it easy to overlay player positions, passes, and shot attempts.

- **Simplifies overlaying player tracking and event data**
  Useful for analyzing movement patterns in soccer, basketball, and other sports.

3.4.4 Plotly (Plotly Technologies Inc. (n.d.))

- **Powerful for interactive and web-based visualizations**
  Users can zoom, pan, and hover over data points for deeper analysis.

- **Supports zooming, hovering, and dynamic updates**
  Especially useful for exploring large datasets.

- **Ideal for real-time data analysis and dashboards**
  Frequently used in dashboards for live game statistics.

3.4.5 Plotnine (Plotnine Developers (n.d.))

- **Python's implementation of the ggplot2 grammar of graphics**
  Provides a structured, layered approach to visualization.

- **Uses a layered approach for building plots**
  Each layer represents a component of the visualization, allowing for flexible customization.

- **Ideal for creating complex visualizations with minimal code**
  Reduces the need for long, detailed Matplotlib code.

4 Matplotlib

4.1 Introduction to Matplotlib

- **Developed by** neurobiologist John D. Hunter in 2003 as a tool for scientific computing in Python.
- **Initially designed** to provide MATLAB-like plotting capabilities.
- **Evolved into** one of the most widely-used plotting libraries, forming the foundation of Python's visualization ecosystem.
- **Used extensively** in scientific computing, data analysis, machine learning, and engineering applications.
- **Supports diverse output formats** such as PNG, PDF, SVG, and interactive backends.

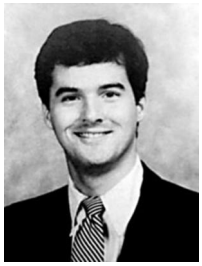
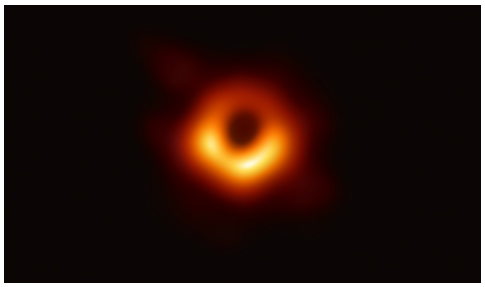


Figure 4.1: Dr. John Hunter; Image courtesy of Princeton Alumni Weekly (n.d.)

Fun Facts

- **Matplotlib was used by NASA** for data visualization during the 2008 landing of the Phoenix spacecraft on Mars.
- **In 2018**, `matplotlib` contributed to generating the first-ever image of a black hole by the Event Horizon Telescope team.



Source: Matplotlib Development Team (2024)

4.2 Pros and Cons of Matplotlib

4.2.1 Strengths

- **Highly customizable** – Enables fine-tuned, publication-quality visualizations.
- **Seamless integration** with **NumPy, Pandas, and SciPy** for scientific computing.
- **Supports multiple output formats** (PNG, PDF, SVG, etc.), making it ideal for diverse use cases.

- **Great for static plots** used in research papers, reports, and presentations.
- **Large community support** with extensive documentation and active development.

4.2.2 Weaknesses

- **Verbose syntax** – Requires more lines of code compared to modern high-level libraries like Seaborn and Plotly.
- **Limited interactivity** – Less suited for interactive dashboards compared to libraries like Plotly.
- **Steeper learning curve** for advanced customizations and handling multiple subplots.
- **Performance bottlenecks** when handling extremely large datasets compared to specialized visualization libraries.

4.3 Installation

Latest available version: 3.10

Install using `pip` (Recommended)

```
pip install matplotlib
```

Install using `conda` (For Anaconda/Miniconda users)

```
conda install matplotlib
```

Importing Matplotlib

```
import matplotlib.pyplot as plt
```

4.4 Anatomy of a Matplotlib Figure

- **Figure**: The entire canvas that contains all plots.
- **Axes**: The main plotting area where data visualization occurs.
- **Axis**: The x-axis and y-axis components.
- **Ticks & Labels**: Marks and corresponding text labels along the axes.
- **Legend**: Explains colors, lines, or markers.
- **Title & Labels**: Describe the overall plot and individual axes.
- **Grid**: Optional background reference lines for better readability.

4.5 Basic Plotting Commands

```
import matplotlib.pyplot as plt
```

- **Line Plot** → `plt.plot(x, y)`
  - Customize with `color`, `linestyle`, `marker`, `label`, `alpha`, `linewidth`
- **Scatter Plot** → `plt.scatter(x, y)`
  - Options: `color`, `marker`, `s` (marker size), `c` (marker color), `alpha`

- **Bar Chart** → `plt.bar(x, y)`
  - Modify with `height`, `width`, `color`, `align`
- **Histogram** → `plt.hist(data, bins=10)`
  - Control `bins`, `range`, `density`, `color`, `alpha`
- **Pie Chart** → `plt.pie(sizes, labels=labels)`
  - Options: `sizes`, `labels`, `colors`, `startangle`, `autopct`

Table 1: Common Matplotlib Markers and Linestyles

Marker	Description	Line Style	Description
`.`	Point	`-`	Solid
`o`	Circle	`--`	Dashed
`v`	Triangle Down	`-.`	Dash-dot
`^`	Triangle Up	`:`	Dotted
`<`	Triangle Left	None	No line
`>`	Triangle Right		
`s`	Square		
`p`	Pentagon		
`*`	Star		
`+`	Plus		
`x`	Cross		
`D`	Diamond		
`h`	Hexagon1		
`H`	Hexagon2		

More Linestyles

More Markers

4.6 Customization Commands

- **Axis Labels** → `plt.xlabel("X-axis Label")`, `plt.ylabel("Y-axis Label")`
  - Customize with `fontsize`, `color`
- **Title** → `plt.title("Plot Title")`
  - Adjust positioning with `left`, `right`, `bottom`, `top`
- **Axis Limits** → `plt.xlim(min, max)`, `plt.ylim(min, max)`
- **Custom Tick Labels** → `plt.xticks(ticks, labels)`, `plt.yticks(ticks, labels)`
- **Grid** → `plt.grid(True, linestyle='--')`
- **Legend** → `plt.legend()`
  - Parameters: `loc`, `fontsize`, `title`, `frameon`, `bbox_to_anchor`

4.7 Multiple Plots & Subplots

- **Create a new figure** → `plt.figure(figsize=(width, height))`
- **Subplots using the functional API** → `plt.subplot(rows, cols, index)`
- **Object-oriented approach** → `fig, ax = plt.subplots()`
  - Supports `nrows`, `ncols`, `figsize`, `sharex`, `sharey`
- **Plot using axes objects** → `ax.plot(x, y)`

4.8 Saving & Displaying Plots

- **Save Figure as an Image** → `plt.savefig("plot.png", dpi=300)`

  - Options: `filename`, `dpi`, `format`

- **Show the Plot** → `plt.show()`
- **Close the Current Figure** → `plt.close()`

5 Example 1: Standard Normal Distribution

Let's plot the standard normal distribution $N(\mu = 1, \sigma = 1)$ pdf given below.

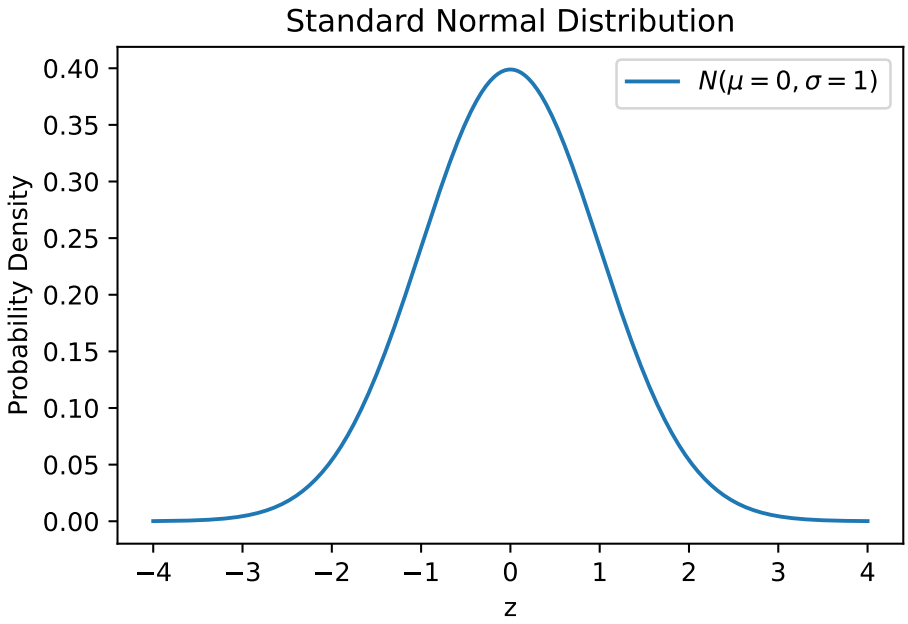
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$$

5.1 Plotting a Fuction

- **Line plot** → `plt.plot(x, y, color , linestyle, marker, label, alpha, linewidth)`

- **Axis labels** → `plt.xlabel("X-axis Label", label, fontsize, color)`, `plt.ylabel("Y-axis Label", label, fontsize, color)`
- **Title** → `plt.title("Plot Title", left, right, bottom, top)`
- **Add legend** → `plt.legend(loc, fontsize, title, frameon, bbox_to_anchor)`
- **Display plot** → `plt.show()`

```
import matplotlib.pyplot as plt
import numpy as np

z = np.linspace(-4,4,1000) # range of z values in plot

# Fuction returns the standard normal pdf
# Input is list/range of z values
# Output is list of computed values along the curve
def stand_norm(z):
    return 1/np.sqrt(2*np.pi)*np.exp(-z**2/2)


f = stand_norm(z) # Standard Normal Distribution from -4 < z < 4

plt.plot(z,f,label=r'$N(\mu=0,\sigma=1)$') # Line Plot
plt.title('Standard Normal Distribution') # Title
plt.xlabel('z') # x label
plt.ylabel('Probability Density') # y label
plt.legend() # show legend


plt.show() # show figure
```



5.1.1 Coloring an area

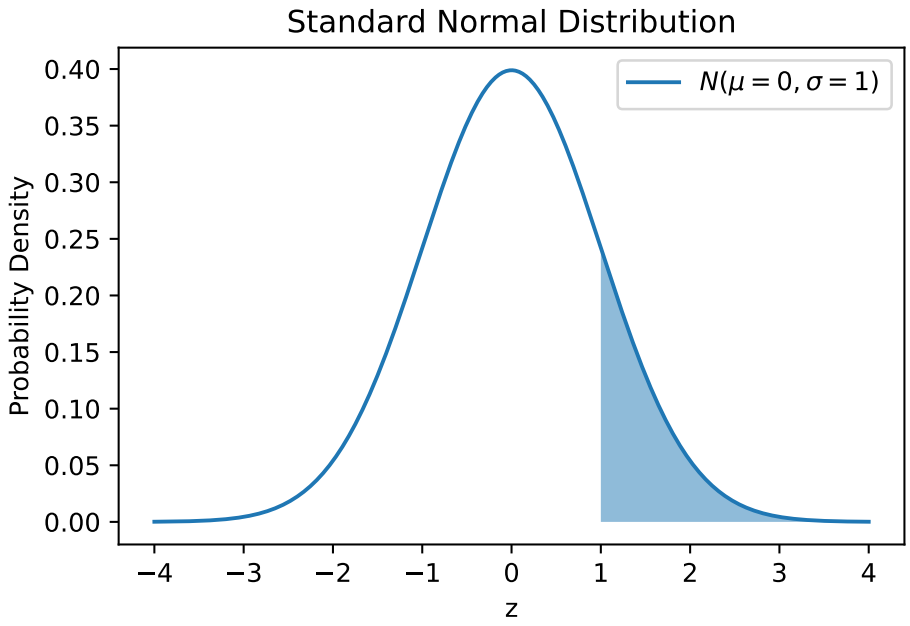
The area under a statistical distribution, particularly the normal distribution, plays a crucial role in probability theory and statistical analysis. It represents cumulative probability, which helps determine the likelihood of a random variable falling within a specific range.

Let's fill the area between $z = 1$ and $z = 4$.

- **Fill area** → `plt.fill_between(x, y1, y2, color, alpha)`

```
plt.plot(z,f,label=r'$N(\mu=0,\sigma=1)$')
plt.title('Standard Normal Distribution')
plt.xlabel('z')
```

```
plt.ylabel('Probability Density')
plt.legend()


z1 = np.linspace(1,4,1000) # z1 defined from 1 to 4 with 1000 values
f1 = stand_norm(z1) # Standard Normal Distribution from 1 < z < 4

plt.fill_between(z1,0,f1,alpha=0.5)

plt.show()
```

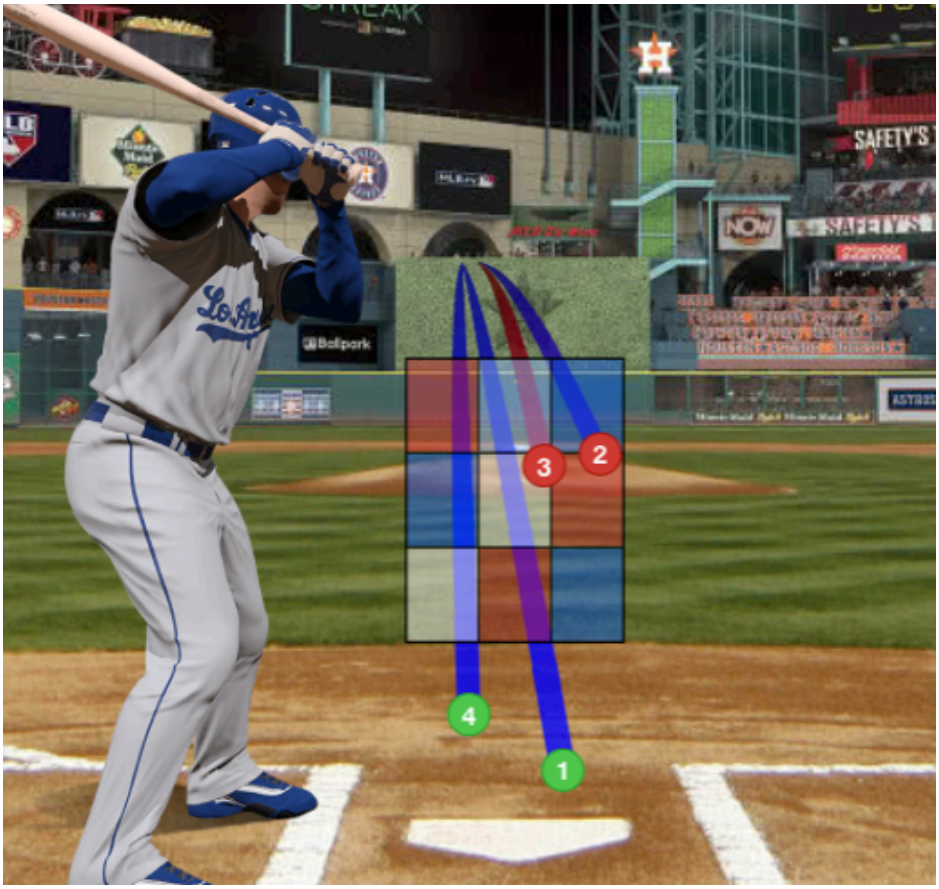


6 Example 2: An example from baseball

Area over home plate is divided into a a grid for pitch analysis.

This grid helps analyze pitch effectiveness, swing decisions, and strike zone control.

- **Heart Zone**: Middle of the strike zone, where hitters make solid contact.
- **Strike Zone**: The legal strike area from the batter's knees to the midpoint of their torso.
- **Shadow Zone**: Just outside the strike zone, where borderline calls happen.
- **Chase Zone**: Further outside, where batters often chase bad pitches.
- **Waste Zone**: Well outside the plate, used for setting up hitters or avoiding damage.



6.1 Baseball Savant

Hosted by MLB, Baseball Savant provides advanced analytics and visualizations using Statcast data.

A go-to resource for analysts, coaches, and fans to explore performance beyond traditional stats.

Key Features

- Statcast tracking for pitches, batted balls, and player movement
- Downloadable CSV files with pitch velocity, spin rate, exit velocity, launch angle, and more
- Interactive visualizations like spray charts, heat maps, and pitch tunnels
- Leaderboards for hitters, pitchers, and fielders based on advanced metrics
- Useful for player comparisons, scouting, and trend analysis

This example with use statcast data obtained with the pybaseball package.

(Source: MLB Advanced Media (2024))

6.2 Aaron Judge's Statcast Data

Star outfielder for the New York Yankees, known for his power hitting and elite defense.

Key Stats & Achievements

- 2024 & 2022 American League Most Valuable Player
- Multiple-time All-Star and Silver Slugger winner
- Career HR leader among active Yankees
- Known for exceptional exit velocity and plate discipline

Let's get Aaron Judge's data for the 2024 Season.

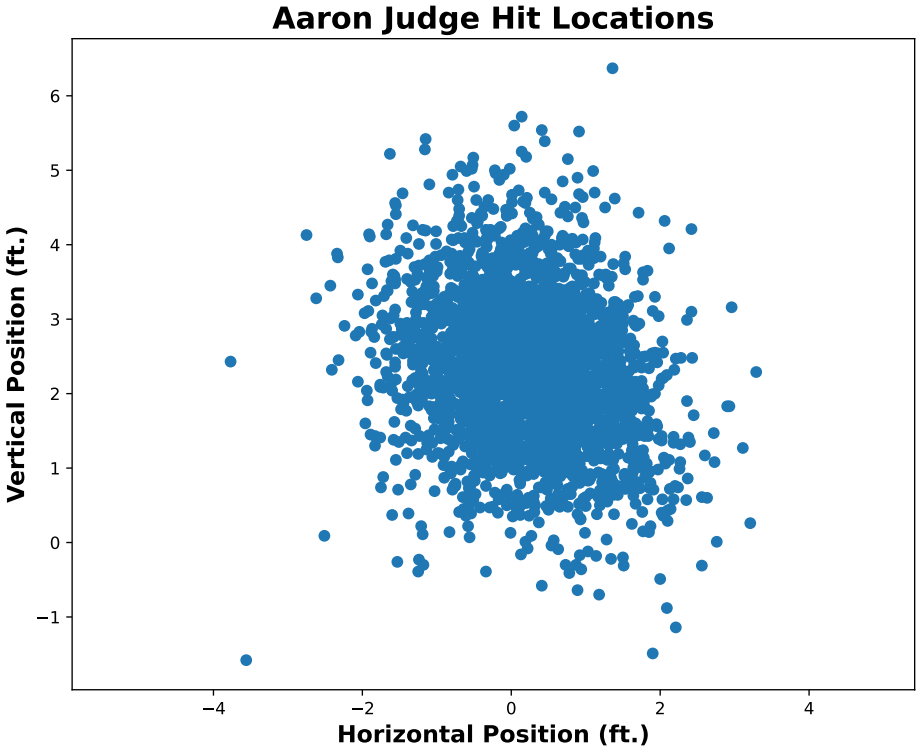
```
import pybaseball as pyball

judge_data = pyball.statcast_batter(start_dt='2024-03-28',end_dt='2024-09-29
platex = judge_data['plate_x']
platez = judge_data['plate_z']
judge_data.head()
```

	pitch_type	game_date	release_speed	release_pos_x	release_pos_z	player_na
0	FF	2024-09-28	94.9	-0.03	6.40	Judge, Aa
1	CH	2024-09-28	86.5	-0.55	6.48	Judge, Aa
2	FF	2024-09-28	94.7	-0.08	6.43	Judge, Aa
3	SL	2024-09-28	87.5	-0.49	6.34	Judge, Aa
4	CU	2024-09-28	79.4	-0.40	6.36	Judge, Aa

6.3 Scatter Plot of Hit Locations

Let's make a scatter plot Judge's hit locations over the home plate.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(9,7))

plt.scatter(platex,platez)
plt.axis('equal')
plt.title('Aaron Judge Hit Locations',weight='bold',fontsize=18)
plt.xlabel('Horizontal Position (ft.)',weight='bold',fontsize=14)
plt.ylabel('Vertical Position (ft.)',weight='bold',fontsize=14)
plt.show()
```

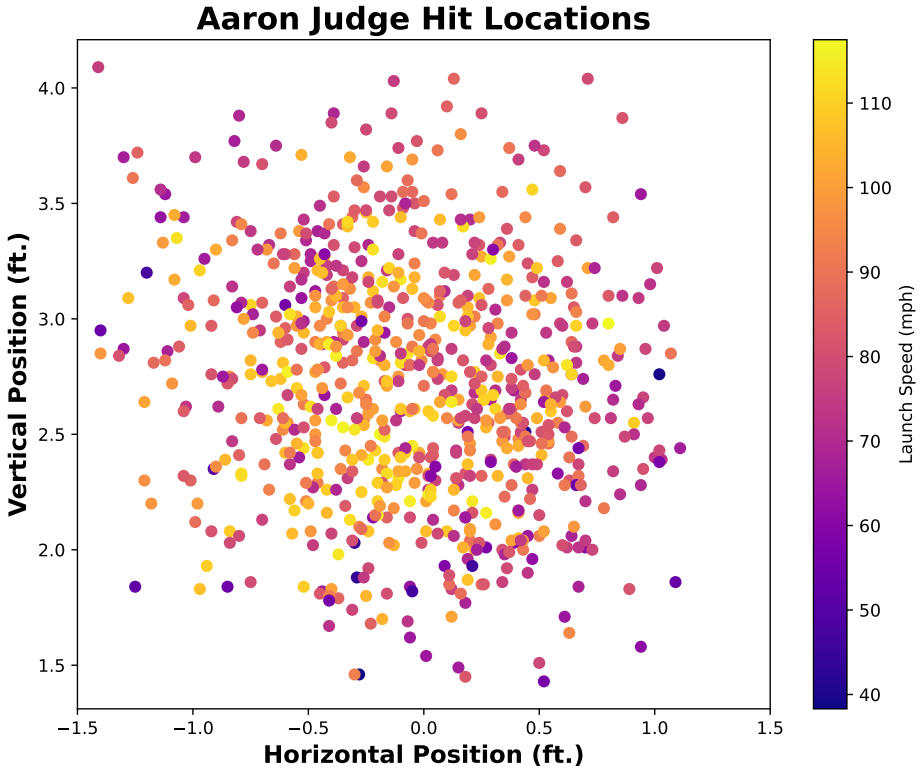


6.4 Hit Location Colored by Exit Velocity

Let's make a scatter plot Judge's hit locations over the home plate colored by his exit velocity.

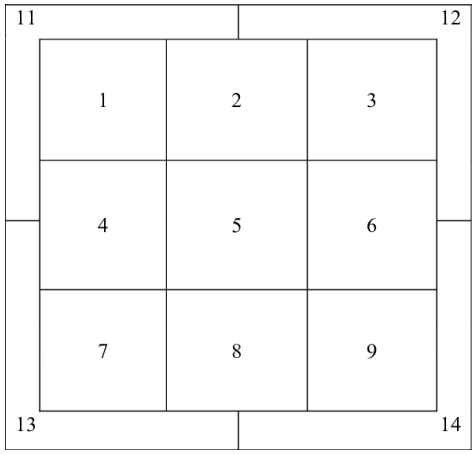
```
plt.figure(figsize=(9,7))
plt.scatter(platex,platez,c=judge_data['launch_speed'],cmap='plasma')
plt.colorbar(label='Launch Speed (mph)')
plt.axis('equal')
```

```
plt.title('Aaron Judge Hit Locations',weight='bold',fontsize=18)
plt.xlabel('Horizontal Position (ft.)',weight='bold',fontsize=14)
plt.ylabel('Vertical Position (ft.)',weight='bold',fontsize=14)
plt.xlim(-1.5,1.5)
plt.show()
```



6.5 Aaron Judge's Hit Locations by Zone

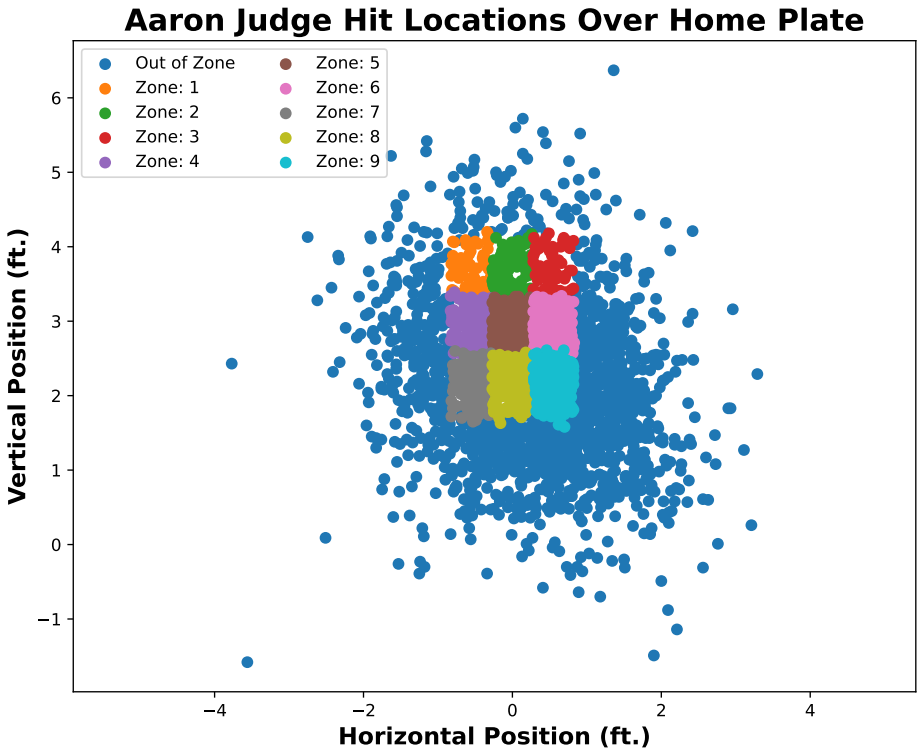
Statcast categorizes the hit location on a 9x9 grid. Each grid has a unique zone number. A ball out side the grid is out of zone and in the strike zone. Here is how the grid is defined.



Let's make a scatter plot Judge's hit locations over the home plate colered by the zones.

```
judge_data['in_zone'] = judge_data['zone'].apply(lambda x: f'Zone: {str(int(x))}' if x in ra

plt.figure(figsize=(9,7))

for category, group in judge_data.groupby('in_zone'):
    plt.scatter(group['plate_x'],group['plate_z'],label=category)

plt.axis('equal')

plt.title('Aaron Judge Hit Locations Over Home Plate',weight='bold',fontsize=18)
```

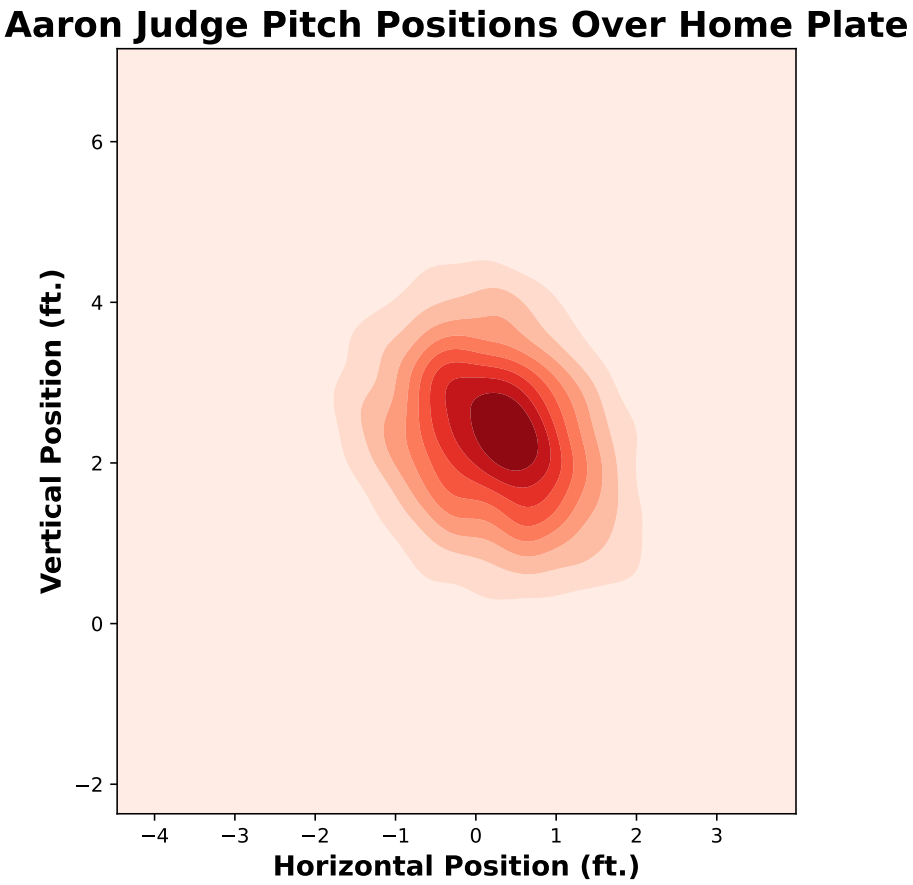
```
plt.xlabel('Horizontal Position (ft.)',weight='bold',fontsize=14)
plt.ylabel('Vertical Position (ft.)',weight='bold',fontsize=14)
plt.legend(ncols=2)
plt.show()
```



6.6 Kernel Density Estimate - Pitch Locations

In the plot above, it is difficult to visualize where majority of the balls are landing. A 2D Kernel Density Estimate (KDE) plot visualizes the probability density of two continuous variables by smoothing data points using a kernel function, typically represented with contour lines or a heatmap. In Seaborn, a Python data visualization library, this can be easily created using `sns.kdeplot()`, helping to identify clusters, relationships, and density patterns in bivariate data.

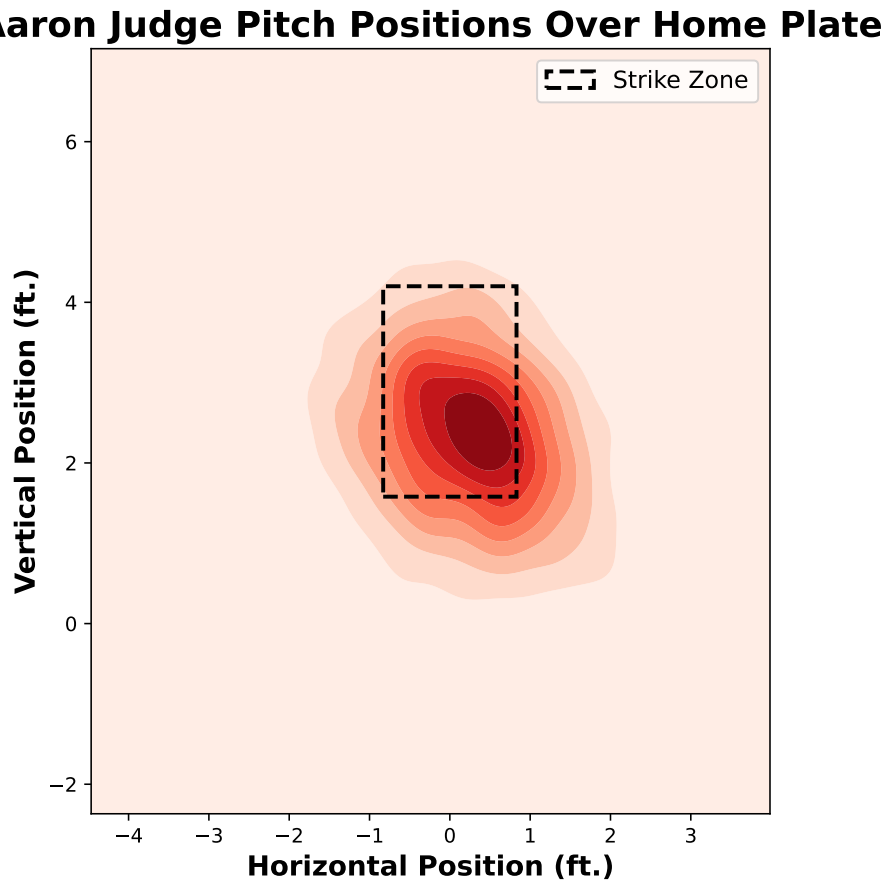
```
import seaborn as sns

plt.figure(figsize=(9,7))

sns.kdeplot(data=judge_data,x='plate_x',y='plate_z',cmap="Reds",
    fill=True,
    thresh=0)

plt.axis('scaled')
plt.title('Aaron Judge Pitch Positions Over Home Plate',weight='bold',fontsize=18)
plt.xlabel('Horizontal Position (ft.)',weight='bold',fontsize=14)
plt.ylabel('Vertical Position (ft.)',weight='bold',fontsize=14)
plt.show()
```



6.6.1 Adding a Strike Zone Box

```
import matplotlib.patches as patches

fig, ax = plt.subplots(figsize=(9,7))
```

```
sns.kdeplot(data=judge_data,x='plate_x',y='plate_z',cmap="Reds",
    fill=True,
    thresh=0)

ax.set_title('Aaron Judge Pitch Positions Over Home Plate',weight='bold',fontsize=18)
ax.set_xlabel('Horizontal Position (ft.)',weight='bold',fontsize=14)
ax.set_ylabel('Vertical Position (ft.)',weight='bold',fontsize=14)
plt.axis('scaled')


in_zone_data = judge_data[judge_data['in_zone'] != 'Out of Zone']

min_x = in_zone_data['plate_x'].min()
max_x = in_zone_data['plate_x'].max()

min_z = in_zone_data['plate_z'].min()
max_z = in_zone_data['plate_z'].max()

rect = patches.Rectangle((min_x,min_z),max_x-min_x,max_z-min_z,facecolor='none',edgecolor='b

ax.add_patch(rect)

plt.legend(fontsize=12)

plt.show()
```


Aaron Judge Pitch Positions Over Home Plate
Strike Zone
Vertical Position (ft.)
Horizontal Position (ft.)
6
4
2
0
−2
−4
−3
−2
−1
0
1
2
3

7 Example 3: Multiple Plots in Basketball

Let's make a 1x2 array of bar charts of the win percentages of top NBA teams in the 2022-23 and 2023-24 seasons.

7.1 Getting NBA data

We will be obtaining data with the `nba-api` package in python.

7.1.1 NBA API (Unofficial)

- Community-maintained API for accessing NBA stats
- Provides box scores, player stats, game logs, and shot charts
- Available through Python packages like `nba_api`

```
from nba_api.stats.endpoints import LeagueStandings

# Fetch data for each season
data_2023 = LeagueStandings(season='2022-23').get_data_frames()[0]
data_2024 = LeagueStandings(season='2023-24').get_data_frames()[0]

# Compute win percentages
```

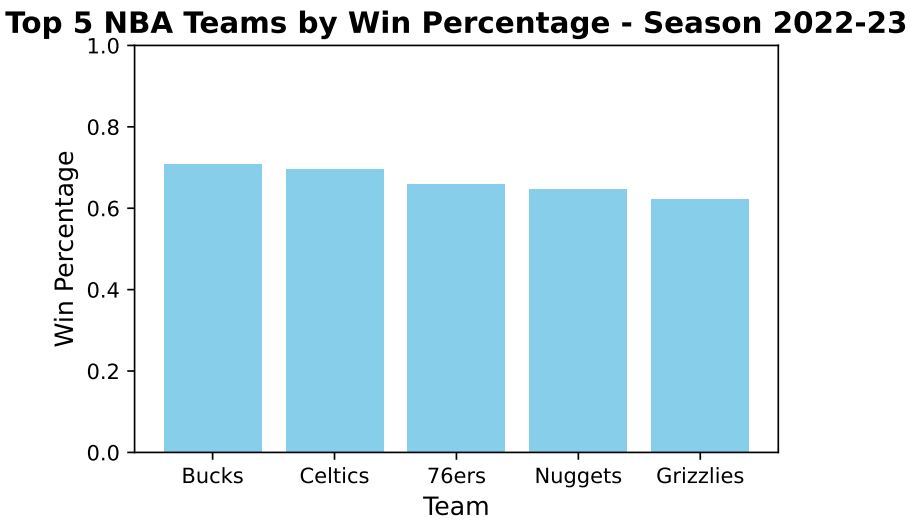
```
data_2023['WinPct'] = data_2023['WINS'] / (data_2023['WINS'] + data_2023['LOS
data_2024['WinPct'] = data_2024['WINS'] / (data_2024['WINS'] + data_2024['LOS

# Get top 5 teams for each season
top_2023 = data_2023.nlargest(5, 'WinPct')[['TeamName','TeamID','WINS','LOSSE
top_2024 = data_2024.nlargest(5, 'WinPct')[['TeamName','TeamID','WINS','LOSSE

team_23 = top_2023['TeamName']
pct_23 = top_2023['WinPct']

team_24 = top_2024['TeamName']
pct_24 = top_2024['WinPct']

print('2022-23 Season','\n',top_2023.head(),'\n')
print('2024-25 Season','\n',top_2024.head())
```

```
2022-23 Season
     TeamName      TeamID  WINS  LOSSES    WinPct
1       Bucks  1610612749    58      24  0.707317
2     Celtics  1610612738    57      25  0.695122
5       76ers  1610612755    54      28  0.658537
0     Nuggets  1610612743    53      29  0.646341
3  Grizzlies  1610612763    51      31  0.621951

2024-25 Season
         TeamName      TeamID  WINS  LOSSES    WinPct
0         Celtics  1610612738    64      18  0.780488
1         Thunder  1610612760    57      25  0.695122
2         Nuggets  1610612743    57      25  0.695122
4  Timberwolves  1610612750    56      26  0.682927
6      Clippers  1610612746    51      31  0.621951
```

7.2 Single Bar Chart

First, let's make a single bar chart.

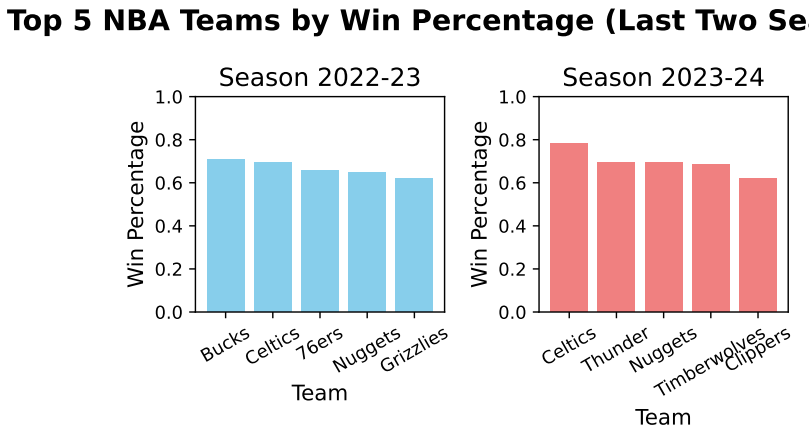
```
import matplotlib.pyplot as plt

plt.bar(team_23, pct_23, color='skyblue')
plt.title("Top 5 NBA Teams by Win Percentage - Season 2022-23", fontsize=14,weight='bold')
plt.xlabel("Team", fontsize=12)
plt.ylabel("Win Percentage", fontsize=12)
plt.ylim(0, 1)
plt.show()
```



7.3 Multiple Bar chart using `plt.subplot`

`plt.subplot(rows, cols, index)` creates a single subplot at the specified position within a grid of rows × cols subplots, with index determining which subplot is active

```
plt.suptitle("Top 5 NBA Teams by Win Percentage (Last Two Seasons)", fontsize

# Plot 2022-23 season
plt.subplot(1, 2, 1)
plt.bar(team_23, pct_23, color='skyblue')
plt.title("Season 2022-23", fontsize=14)
plt.xlabel("Team", fontsize=12)
plt.ylabel("Win Percentage", fontsize=12)
plt.ylim(0, 1)
plt.xticks(rotation=30)

# Plot 2023-24 season
plt.subplot(1, 2, 2)
plt.bar(team_24, pct_24, color='lightcoral')
plt.title("Season 2023-24", fontsize=14)
plt.xlabel("Team", fontsize=12)
plt.ylabel("Win Percentage", fontsize=12)
plt.ylim(0, 1)
plt.xticks(rotation=30)

# Adjust layout and show plot
plt.tight_layout()
plt.show()
```

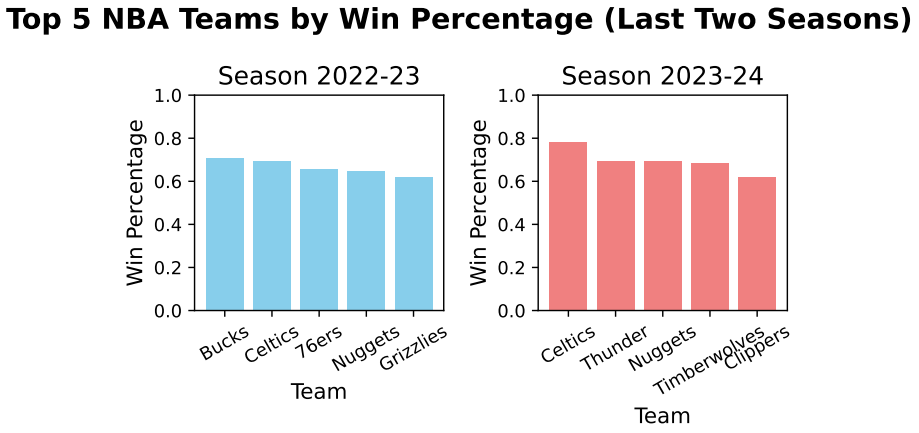


7.4 Multiple Bar chart using `plt.subplots`

`plt.subplots(rows, cols)` creates a figure and an array of subplot axes in one call, allowing for easier iteration and direct axis manipulation. This method allows for additional customization.

```
fig, axes = plt.subplots(1, 2)
fig.suptitle("Top 5 NBA Teams by Win Percentage (Last Two Seasons)", fontsize=16, fontweight

# Plot 2022-23 season
axes[0].bar(team_23, pct_23, color='skyblue')
axes[0].set_title("Season 2022-23", fontsize=14)
axes[0].set_xlabel("Team", fontsize=12)
axes[0].set_ylabel("Win Percentage", fontsize=12)
axes[0].set_ylim(0, 1)
axes[0].tick_params(axis='x', rotation=30)

# Plot 2023-24 season
axes[1].bar(team_24, pct_24, color='lightcoral')
```

```
axes[1].set_title("Season 2023-24", fontsize=14)
axes[1].set_xlabel("Team", fontsize=12)
axes[1].set_ylabel("Win Percentage", fontsize=12)
axes[1].set_ylim(0, 1)
axes[1].tick_params(axis='x', rotation=30)

# Adjust layout and show plot
plt.tight_layout()  # Adjust for title spacing
plt.show()
```



8 Example 4: An example from Formula One

8.1 Formula 1 Data Sources

Formula One data is available through a few different sources.

- **FastF1 (Source: FastF1 Development Team (2024))**
  - Python library for accessing and analyzing F1 data
  - Provides lap times, telemetry, sector data, and tire strategies
  - Useful for race strategy analysis and driver performance comparison
- **Ergast API (Source: Ergast (2024))**
  - Free API with historical and current F1 data
  - Includes race results, driver standings, and circuit information
  - Good for building interactive dashboards and statistical models
- **FIA Website & Documents (Source: Fédération Internationale de l'Automobile (FIA) (2024))**
  - Official source for race reports, technical regulations, and timing sheets

- Provides PDFs and CSVs with timing and classification data
- Ideal for detailed race analysis and rule interpretation

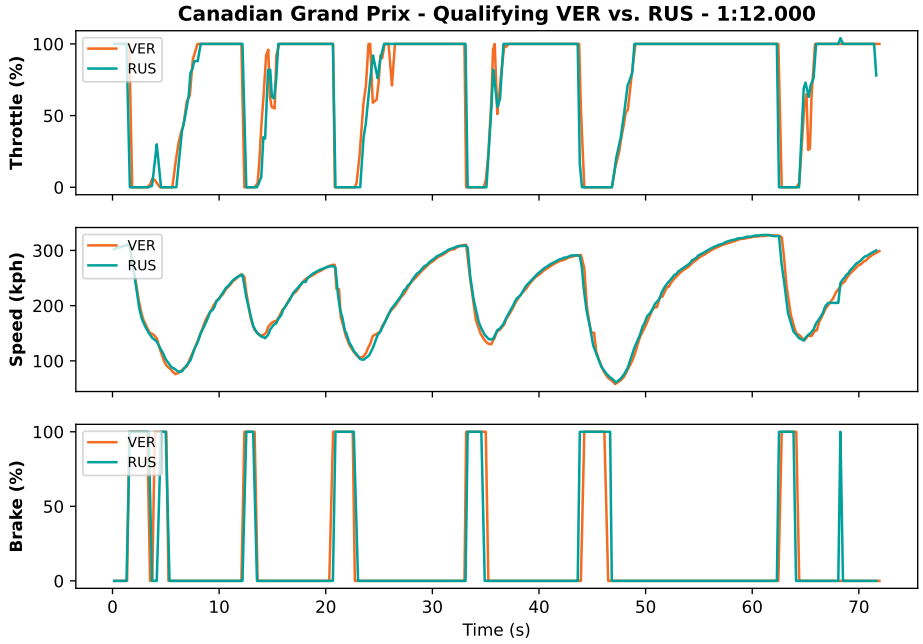
We will be using the `fastf1` package to get our data.

8.2 Telemetry

During the 2024 Canadian Grand Prix Qualifying, RedBull driver Max Verstappen and Mercedes driver George Russel set the exact same time of 1:12:000. Let's visualize there use of throttle, brake, and speed throughout their lap.

```
import fastf1
import matplotlib.pyplot as plt

fastf1.set_log_level('CRITICAL')

quali = fastf1.get_session(2024,'Canada','Q')

quali.load()

ver_lap = quali.laps.pick_drivers('VER').pick_fastest().get_car_data()
rus_lap =  quali.laps.pick_drivers('RUS').pick_fastest().get_car_data()

fig, ax = plt.subplots(nrows=3,sharex = True,figsize=(9,6))
ax[0].plot(ver_lap['Time'].dt.total_seconds(),ver_lap['Throttle'], label='VE
ax[0].plot(rus_lap['Time'].dt.total_seconds(),rus_lap['Throttle'],label='RUS
ax[0].set_ylabel('Throttle (%)',weight='bold')
ax[0].legend(prop={'size':'small'},loc='upper left')

ax[1].plot(ver_lap['Time'].dt.total_seconds(),ver_lap['Speed'],label='VER',c
```

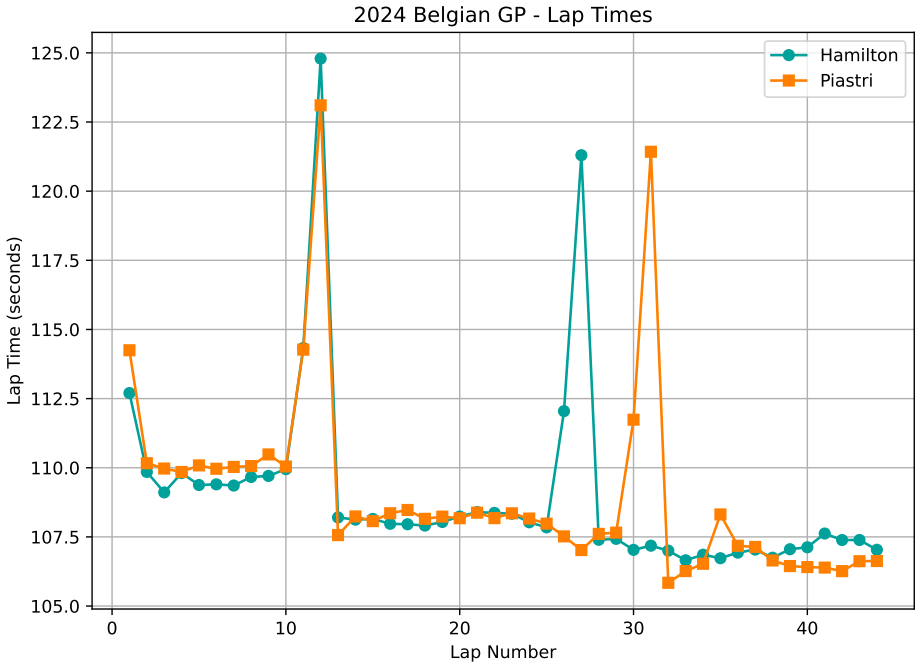
```
ax[1].plot(rus_lap['Time'].dt.total_seconds(),rus_lap['Speed'],label='RUS',color='#00A19B')
ax[1].set_ylabel('Speed (kph)',weight='bold')
ax[1].legend(prop={'size':'small'},loc='upper left')

ax[2].plot(ver_lap['Time'].dt.total_seconds(),100*ver_lap['Brake'],label='VER',color= '#F360
ax[2].plot(rus_lap['Time'].dt.total_seconds(),100*rus_lap['Brake'],label='RUS',color='#00A19
ax[2].set_ylabel('Brake (%)',weight='bold')
ax[2].legend(prop={'size':'small'},loc='upper left')

ax[0].set_title('Canadian Grand Prix - Qualifying VER vs. RUS - 1:12.000',weight='bold')
plt.xlabel('Time (s)')
plt.show()
```



8.3 Lap Times over Race Distance

During the 2024 Belgian Grand Prix, race winner Lewis Hamilton finished 0.647s ahead of Oscar Piastri. Let's visualize their laptimes over the race distance.

```
# Load the session
year, gp, session_type = 2024, "Belgian", "R"

session = fastf1.get_session(year, gp, session_type)
session.load()

# Get lap times for Hamilton and Piastri
drivers = {"HAM": "Lewis Hamilton", "OSC": "Oscar Piastri"}
laps_ham = session.laps.pick_drivers("HAM")
laps_ver = session.laps.pick_drivers("PIA")

# Plot
plt.figure(figsize=(8.5,6))
plt.plot(laps_ham["LapNumber"], laps_ham["LapTime"].dt.total_seconds(), label
plt.plot(laps_ver["LapNumber"], laps_ver["LapTime"].dt.total_seconds(), label

plt.xlabel("Lap Number")
plt.ylabel("Lap Time (seconds)")
plt.title(f"{year} {gp} GP - Lap Times")
plt.legend()
plt.grid(True)
plt.show()
```


2024 Belgian GP - Lap Times
Hamilton
Piastri
Lap Time (seconds)
Lap Number
125.0
122.5
120.0
117.5
115.0
112.5
110.0
107.5
105.0
0
10
20
30
40

9 Advanced Examples

Here are some advanced examples of visualizing sports data with `matplotlib`.

9.1 Baseball Spray Chart

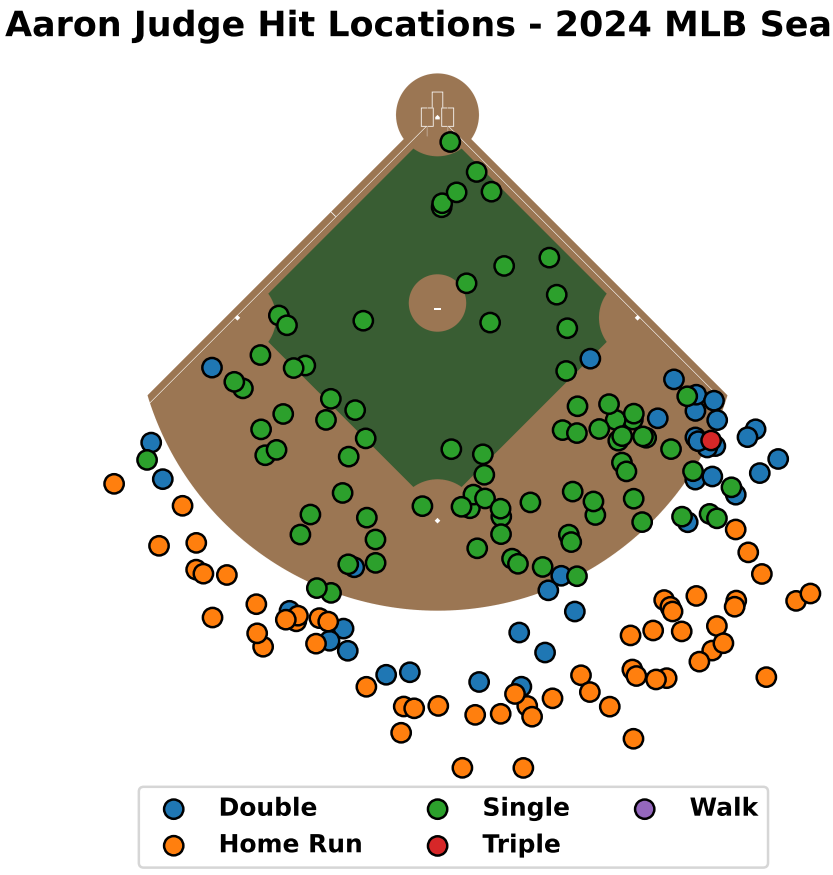
In baseball, the ball location on the field can reveal pattern's in the strengths and weakness of a batter or a team. Let's make a spray chart for Aaron Judge's 2023-24 MLB season.

Sportypy is a python package built on matplotlib to add courts and fields to sports visualizations. We'll be using this package to add a baseball pitch to our plot background.

```
from sportypy.surfaces.baseball import MLBField
import pybaseball as pyball
import matplotlib.pyplot as plt

data = pyball.statcast_batter(start_dt='2024-03-28',end_dt='2024-09-29',player_id=592450)

data = data.groupby('events')

fig, ax = plt.subplots(figsize=(8,6))

field = MLBField(x_trans=130,y_trans=-213,rotation=180)
```

```
field.draw(ax,xlim=(0,250))

for i in data.groups.keys():

    hcx = data.get_group(i)['hc_x']
    hcy = -1*data.get_group(i)['hc_y']

    if i in ['home_run','double','triple','walk','single']:
        label = ' '.join([x.capitalize() for x in i.split('_')])
        field.scatter(hcx,hcy,label=label,linewidths=1,edgecolor='black',s=60

plt.ylim(-240,20)

plt.legend(ncols=3,prop={'weight':'bold','size':'medium'},loc='lower center')
plt.title('Aaron Judge Hit Locations - 2024 MLB Season',weight='bold',fontsiz
plt.show()
```

```
Gathering Player Data
```

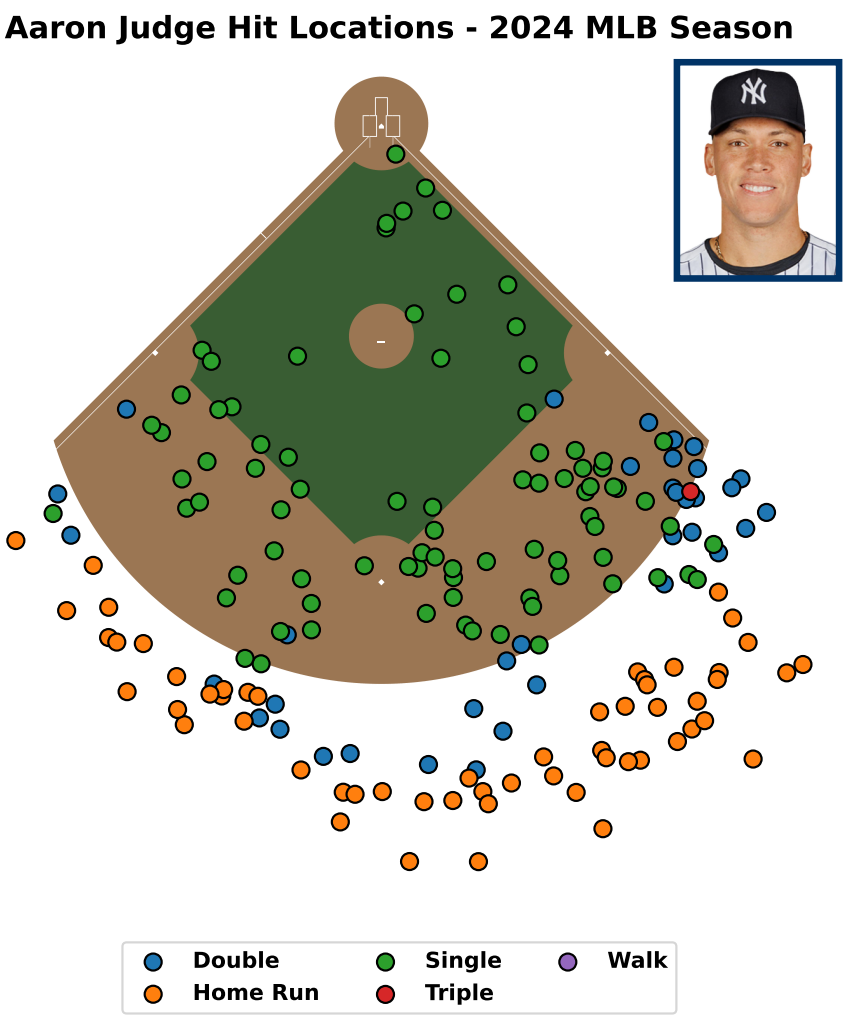


9.1.1 Adding Player Headshot to a Plot

```
import PIL.Image
import matplotlib.patches as patches

data = pyball.statcast_batter(start_dt='2024-03-28',end_dt='2024-09-29',player_id=592450)
```

```
data = data.groupby('events')

fig, ax = plt.subplots(figsize=(8,8))

field = MLBField(x_trans=130,y_trans=-213,rotation=180)

field.draw(ax,xlim=(0,250))


for i in data.groups.keys():

    hcx = data.get_group(i)['hc_x']
    hcy = -1*data.get_group(i)['hc_y']

    if i in ['home_run','double','triple','walk','single']:
        label = ' '.join([x.capitalize() for x in i.split('_')])
        field.scatter(hcx,hcy,label=label,linewidths=1,edgecolor='black',s=60


plt.ylim(-250,20)

# Add Headshot
headshot = PIL.Image.open('../images/aaron_judge headshot.png')

plt.imshow(headshot,extent=(82,127,-43,17))
rect = patches.Rectangle((82, -43), 45, 60, linewidth=3, edgecolor='#003366'
ax.add_patch(rect)

plt.legend(ncols=3,prop={'weight':'bold','size':'medium'},loc='lower center'
plt.title('Aaron Judge Hit Locations - 2024 MLB Season',weight='bold',fontsiz
plt.show()
```

```
Gathering Player Data
```



9.2 Basketball Heatmap

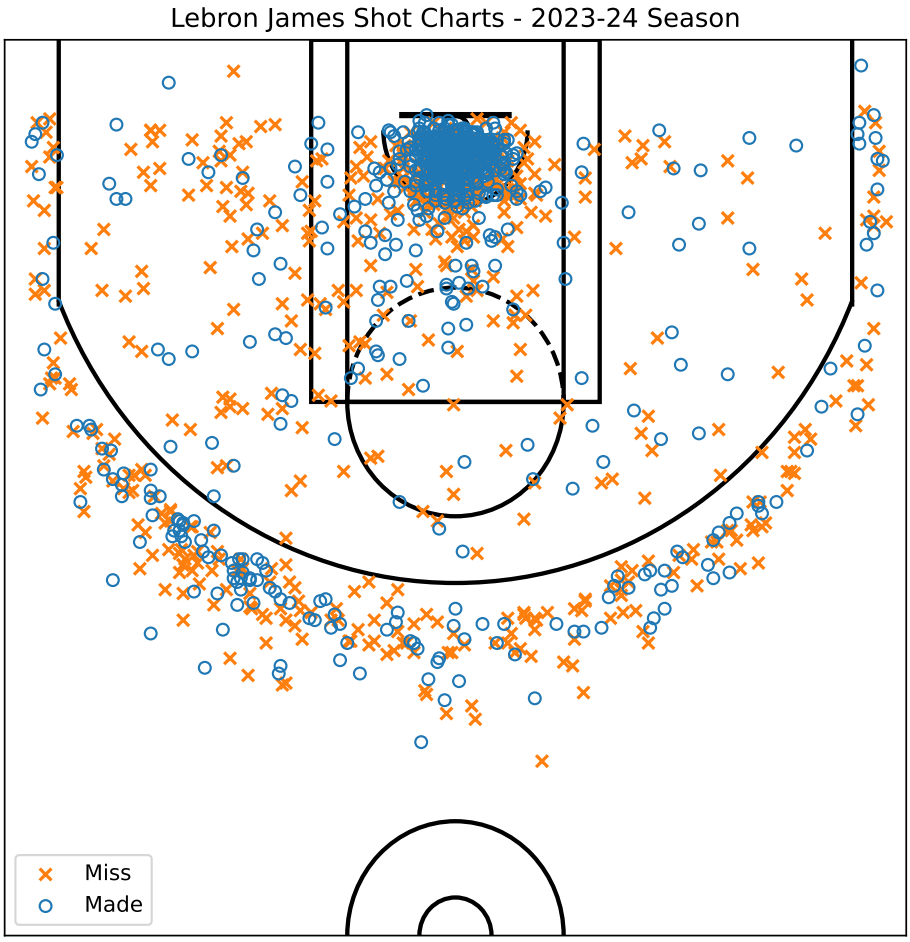
A basketball player's shot chart is a visual representation of their shooting performance across different areas of the court. By plotting each shot attempt, it highlights the player's shooting efficiency, preferred spots, and areas of improvement. This data can be used to analyze shooting consistency, identify high-percentage zones, and strategize for both offensive and defensive game planning.

Let's make one for Lebron James!

```
from nba_api.stats.endpoints import shotchartdetail
from matplotlib.patches import Circle, Rectangle, Arc

lebron_data = shotchartdetail.ShotChartDetail(team_id = 0, player_id=2544,sea

fig, ax = plt.subplots(figsize=(7.5,7.5))


def draw_court(ax=None, color='black', lw=2, outer_lines=False):

    if ax is None:
        ax = plt.gca()

    hoop = Circle((0, 0), radius=7.5, linewidth=lw, color=color, fill=False)

    backboard = Rectangle((-30, -7.5), 60, -1, linewidth=lw, color=color)

    outer_box = Rectangle((-80, -47.5), 160, 190, linewidth=lw, color=color,

    inner_box = Rectangle((-60, -47.5), 120, 190, linewidth=lw, color=color,

    # Create free throw top arc
    top_free_throw = Arc((0, 142.5), 120, 120, theta1=0, theta2=180,
```

```
                          linewidth=lw, color=color, fill=False)
    # Create free throw bottom arc
    bottom_free_throw = Arc((0, 142.5), 120, 120, theta1=180, theta2=0,
                            linewidth=lw, color=color, linestyle='dashed')
    # Restricted Zone, it is an arc with 4ft radius from center of the hoop
    restricted = Arc((0, 0), 80, 80, theta1=0, theta2=180, linewidth=lw,
                     color=color)

    # Three point line
    # Create the side 3pt lines, they are 14ft long before they begin to arc
    corner_three_a = Rectangle((-220, -47.5), 0, 140, linewidth=lw,color=color)
    corner_three_b = Rectangle((220, -47.5), 0, 140, linewidth=lw, color=color)
    # 3pt arc - center of arc will be the hoop, arc is 23'9" away from hoop
    three_arc = Arc((0, 0), 475, 475, theta1=22, theta2=158, linewidth=lw,color=color)

    center_outer_arc = Arc((0, 422.5), 120, 120, theta1=180, theta2=0,linewidth=lw, color=co
    center_inner_arc = Arc((0, 422.5), 40, 40, theta1=180, theta2=0,linewidth=lw, color=colo
    court_elements = [hoop, backboard, outer_box, inner_box, top_free_throw,
                      bottom_free_throw, restricted, corner_three_a,
                      corner_three_b, three_arc, center_outer_arc,
                      center_inner_arc]

    if outer_lines:
        outer_lines = Rectangle((-250, -47.5), 500, 470, linewidth=lw,color=color, fill=Fals
        court_elements.append(outer_lines)

    for element in court_elements:
        ax.add_patch(element)

    return ax

draw_court(ax)
```

```
grouped = lebron_data.groupby('EVENT_TYPE')
made = grouped.get_group('Made Shot')
miss = grouped.get_group('Missed Shot')
ax.scatter(miss['LOC_X'],miss['LOC_Y'],label='Miss',marker='x',color='tab:ora
ax.scatter(made['LOC_X'],made['LOC_Y'],label='Made',color='none',edgecolor='t
plt.ylim(422.5, -47.5)
plt.xlim(-250,250)
plt.title('Lebron James Shot Charts - 2023-24 Season')
ax.xaxis.set_tick_params(labelbottom=False)
ax.yaxis.set_tick_params(labelleft=False)
ax.set_xticks([])
ax.set_yticks([])
plt.legend()
plt.show()
```



(Source: Tjortjoglou (2024))

10 Animation with Matplotlib

Animating plots is also possible with `matplotlib`. Animated maps and plots are valuable for showing changes over time or across locations, making trends and patterns easier to see. They're useful in fields like public health, where animated maps can show how a disease spreads, or in economics, where plots can track market trends. Environmental scientists also use animated weather maps to illustrate seasonal shifts. These visuals make complex data clearer and help in understanding and decision-making.

The first example and background information was pulled from Into to Data Science - Fall 2024.

10.0.1 Animating Plots

10.0.1.1 Matplotlib's FuncAnimation

```
from matplotlib.animation import FuncAnimation
```

`FuncAnimation` is used to create animations in Matplotlib by repeatedly calling a user-defined function.

`anim = FuncAnimation(fig,func,frames,interval,repeat,repeat_delay)`

Key Inputs

- `fig`: Matplotlib figure object.

- `func`: The update function for each frame.
- `frames`: Sequence or number of frames.
- `interval`: Time interval between frames (ms).
- `repeat`: Whether to repeat animation (True/False).
- `repeat_delay`: Delay before repeating (ms).

Source: Hunter & Matplotlib Development Team (2023)

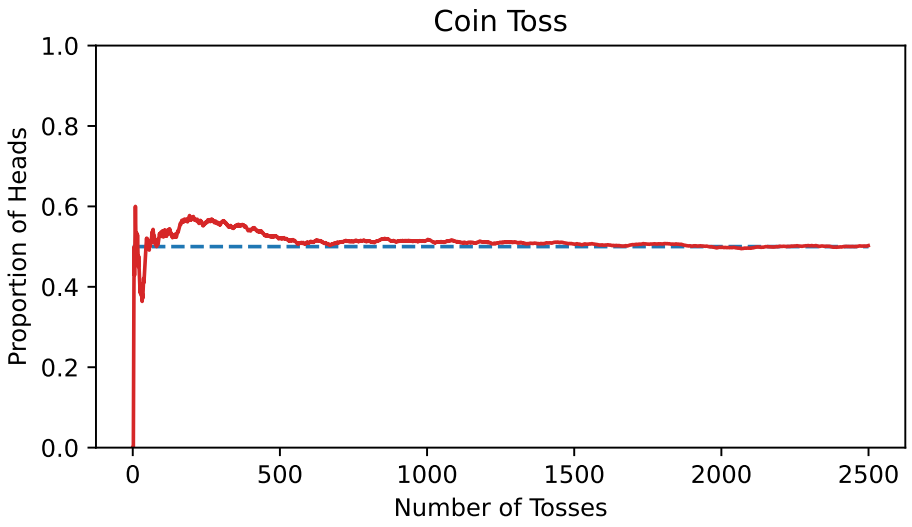
Check out Matplotlib's FunctionAnimation Documentation for more information.

10.1 Coin Toss Example

Let's create a line plot to show how the proportion of heads in coin tosses changes as the number of tosses increases.

```
import random
import matplotlib.pyplot as plt

random.seed(3255)

def prob_heads(trials):
    result = []
    prop_heads = []
    for i in range(trials):
        toss = random.randint(0,1)
        result.append(toss)
        prop_heads.append(sum(result)/len(result))
    return prop_heads

plt.figure(figsize=(6,3))
plt.hlines(0.5,0,2500,linestyles='dashed')
plt.plot(prob_heads(2500),color='tab:red')
```

```
plt.ylim(0,1)
plt.title("Coin Toss")
plt.ylabel('Proportion of Heads')
plt.xlabel('Number of Tosses')
plt.show()
```



10.1.1 Coin Toss Animtion

Using `FuncAnimation`, we can animate the coin toss plot we previously made.

```
prop_heads = prob_heads(2500)

frames = range(len(prop_heads))
```

```
fig, ax = plt.subplots(figsize=(12,6))

def update(frame):
    # Clear previous frame
    ax.clear()

    # Add title, and labels
    ax.set_title('Coin Toss')
    ax.set_ylabel('Proportion of Heads')
    ax.set_xlabel('Number of Tosses')
    ax.set_ylim(0,1)

    # Plot data
    ax.hlines(0.5,0,frame+1,linestyles='dashed')
    ax.plot(range(1,frame+1),prop_heads[:frame],color='tab:red')


anim = FuncAnimation(fig,update,frames=frames,repeat=False)

anim.save('coin_toss.gif',writer='Pillow',fps=50)

plt.show()
```

10.1.2 A Step Further - Coin Toss Animation

We can take this a step further by labeling the current proportion value for each frame.

```
prop_heads = prob_heads(2500)

frames = range(len(prop_heads))
```

```
fig, ax = plt.subplots(figsize=(12,6))

def update(frame):
    ax.clear()
    ax.set_title('Coin Toss')
    ax.set_ylabel('Proportion of Heads')
    ax.set_xlabel('Number of Tosses')
    ax.hlines(0.5,0,frame+1,linestyles='dashed')
    ax.set_ylim(0,1)

    # Add text
    ax.text(frame+1,prop_heads[frame]*1.05,f'{prop_heads[frame]:.3f}',weight='bold')

    ax.plot(range(1,frame+1),prop_heads[:frame],color='tab:red')


anim = FuncAnimation(fig,update,frames=frames)


plt.show()
```

10.2 Bar Chart Animation

Let's animate a bar chart of the cummalitive points scored by the top ten NBA players over the course of a season. Here I am using `plt.barh` to create a horizontal bar chart.

```
import pandas as pd
from nba_api.stats.endpoints import leaguegamelog
from matplotlib.animation import FuncAnimation
import matplotlib.pyplot as plt
```

```
plt.style.use('dark_background')
# Download all games
games = leaguegamelog.LeagueGameLog(season='2023-24',
 season_type_all_star='Regular Season', timeout=60,
 player_or_team_abbreviation='P').get_data_frames()[0]

# Prepare the data
games['GAME_DATE'] = pd.to_datetime(games['GAME_DATE'])

# We want PLAYER_NAME, GAME_DATE, PTS
df = games[['PLAYER_NAME', 'GAME_DATE', 'PTS']]

# Step 3: Sort by date
df = df.sort_values('GAME_DATE')

# Cumulative points per player over time
df['CUM_PTS'] = df.groupby('PLAYER_NAME')['PTS'].cumsum()

# Get top ten players
top_players = df.groupby('PLAYER_NAME')['PTS'].sum().sort_values(
    ascending=False).head(10).index
df = df[df['PLAYER_NAME'].isin(top_players)]


plt.rcParams['font.weight'] = 'bold'

fig, ax = plt.subplots(figsize=(12,8))

def func(x):
    ax.clear()

    data = df[df['GAME_DATE']<=x]
    data = data.groupby('PLAYER_NAME')
```

```
    players = data.groups.keys()

    plot_df = pd.DataFrame(index=players)
    plot_df['PTS']= [data.get_group(x)['CUM_PTS'].max() for x in players]
    plot_df = plot_df.sort_values('PTS')

    names = ['\n'.join(i.split(' ')) for i in plot_df.index.to_list()]

    hbars = ax.barh(names, plot_df['PTS'])

    ax.bar_label(hbars,plot_df['PTS'])

    ax.text(0.98,0.05,f'Date: {x.strftime('%B %d, %Y')}',transform=ax.transAxes,
            weight='bold',fontsize=15, color='white',ha='right',va='bottom')

    ax.set_title('Cummalitive Points Scored by Top Players - 2023-24 NBA Season',weight='bol
    ax.set_xlabel('Points',weight='bold')



anim = FuncAnimation(fig,func,frames=df['GAME_DATE'])

anim.save('animation.gif',writer='Pillow',fps=15)
```

10.3 Saving your Animation

10.3.1 GIF

To save your animation as a **GIF**:

- **Writer**: `Pillow`
- **Command**: Use `anim.save()` with the `writer='Pillow'` option.

```
pip install pillow # pip users
conda install -c conda-forge pillow # conda users
```

Example:

```
anim.save('animation.gif', writer='Pillow', fps=30, dpi=200)
```

10.3.2 MP4

To save your animation as **MP4**:

- **Writer**: `ffmpeg`
- **Command**: Use `anim.save()` with the `writer='ffmpeg'` option.

```
conda install -c conda-forge ffmpeg # conda users
```

Pip Users:

1. Download from ffmeg.org
2. Extract the folder
3. Add the bins folder path to your system variables.

Example:

```
anim.save('animation.mp4', writer='ffmpeg', fps=30, dpi=300)
```

References

Ergast. (2024). *Ergast motor racing data API*. https://ergast.com/mrd/

FastF1 Development Team. (2024). *FastF1 documentation*. https://docs.fastf1.dev/index.html

Fédération Internationale de l'Automobile (FIA). (2024). *2025 FIA formula one world championship*. https://www.fia.com/events/fia-formula-one-world-championship/season-2025/2025-fia-formula-one-world-championship

Hunter, J. D., & Matplotlib Development Team, the. (2023). *Matplotlib.animation.FuncAnimation*. https://matplotlib.org/stable/api/_as_gen/matplotlib.animation.FuncAnimation.html

International Forum of Visual Practitioners. (n.d.). *Why our brain loves pictures*. https://ifvp.org/content/why-our-brain-loves-pictures

Johns Hopkins University. (n.d.). *COVID-19 dashboard by the center for systems science and engineering (CSSE)*. https://coronavirus.jhu.edu/map.html

Matplotlib Development Team. (2024). *Matplotlib: Visualization with python*. https://matplotlib.org/stable/index.html

MLB Advanced Media. (2024). *Statcast search*. https://baseballsavant.mlb.com/statcast_search

Plotly Technologies Inc. (n.d.). *Plotly python graphing library*. https://plotly.com/python/

Plotnine Developers. (n.d.). *Plotnine: A grammar of graphics for python*. https://plotnine.org/

Princeton Alumni Weekly. (n.d.). *John d. Hunter '90 memorial*. https://paw.princeton.edu/memorial/john-d-hunter-90

QuickStart. (2024). *Data analytics and visualization: Revolutionizing sports*. https://www.quickstart.com/blog/data-science/data-analytics-and-visualization-revolutionizing-sports/

Seaborn Development Team. (2024). *Seaborn: Statistical data visualization*. https://seaborn.pydata.org/

SportsDataverse. (2024). *SportyPy: Sports data analysis in python*. https://sportypy.sportsdataverse.org/

Tjortjoglou, S. (2024). *NBA shot charts with python*. http://savvastjortjoglou.com/nba-shot-sharts.html