Defensive Disruption

Esteban Peredo

May 2024

1 Line-by-line comments of the code

```
# Importing pandas, a library providing data structures and data analysis tools
import pandas as pd

# Importing numpy, a library for the Python programming language, adding support for large,
# multi-dimensional arrays and matrices, along with a large collection of high-level
    mathematical
# functions to operate on these arrays.
import numpy as np

# Importing norm and gaussian_kde for statistical analysis and percentileofscore for
    calculating
# percentile scores
from scipy.stats import norm, gaussian_kde, percentileofscore

# Setting the maximum number of columns pandas will display
pd.options.display.max_columns = None

# Importing players and shotchartdetail from nba_api.stats.static and nba_api.stats.endpoints
    respectively.
# These are used to fetch player data and shot chart details from the NBA API
from nba_api.stats.static import players
from nba_api.stats.endpoints import shotchartdetail
from nba_api.stats.endpoints import playercareerstats

# Importing matplotlib.pyplot, a plotting library used for 2D graphics
import matplotlib.pyplot as plt

# Importing seaborn, a Python data visualization library based on matplotlib
import seaborn as sns

# Importing various modules from matplotlib to create and customize the shot chart
from matplotlib import cm
from matplotlib.patches import Circle, Rectangle, Arc, ConnectionPatch
from matplotlib.patches import Polygon
from matplotlib.collections import PatchCollection
from matplotlib.colors import LinearSegmentedColormap, ListedColormap, BoundaryNorm
from matplotlib.path import Path
from matplotlib.patches import PathPatch
import matplotlib.patches as mpatches

# Setting the style of seaborn to 'white'
sns.set_style('white')

# Setting the default color palette of seaborn
sns.set_color_codes()

# Importing necessary libraries and modules
from nba_api.stats.static import teams
from nba_api.stats.endpoints import leaguegamefinder
```

```
import pandas as pd
from functools import wraps
import time

# Defining a decorator function to retry a function call in case of a specified exception
def retry(ExceptionToCheck, tries=4, delay=3, backoff=2, logger=None):
    def deco_retry(f):
        @wraps(f)
        def f_retry(*args, **kwargs):
            mtries, mdelay = tries, delay
            while mtries > 1:
                try:
                    return f(*args, **kwargs) # Try to execute the function
                except ExceptionToCheck as e: # If exception occurs, print a message and wait for
                     some time
                                            # before retrying
                    msg = f"{str(e)}, Retrying in {mdelay} seconds..."
                    if logger:
                        logger.warning(msg)
                    else:
                        print(msg)
                    time.sleep(mdelay)
                    mtries -= 1
                    mdelay *= backoff
            return f(*args, **kwargs) # Final attempt to execute the function
        return f_retry
    return deco_retry

# Creating a retry decorator for API calls
retry_api_call = retry(Exception)

# Fetching a list of all NBA teams
nba_teams = teams.get_teams()

# Initializing a list to store games data for all teams
all_teams_games = []

# Looping over each team
for team in nba_teams:
    team_id = team['id']
    team_abbreviation = team['abbreviation']

    team_games = []

    # Specifying the season for which to fetch data
    season_id = '2023-24'

    # Defining a function to fetch games for a team for a specific season, and decorating it
        with the retry decorator
    @retry_api_call
    def fetch_team_games(team_id, season_id):
        gamefinder = leaguegamefinder.LeagueGameFinder(team_id_nullable=team_id, season_nullable
            =season_id)
        games = gamefinder.get_data_frames()[0]
        return games

    # Trying to fetch games for the current team
    try:
        games = fetch_team_games(team_id, season_id)
        team_games.append(games)
    except Exception as e: # If an exception occurs, print a message and skip to the next team
        print(f"Failed to fetch games for team {team_abbreviation}: {e}")
        continue
    # Concatenating all the dataframes for the current team into a single dataframe
    team_games_df = pd.concat(team_games, ignore_index=True)

    # Adding a column for team abbreviation
    team_games_df['TEAM_ABBREVIATION'] = team_abbreviation

    # Appending the games for the current team to the list of all teams' games
    all_teams_games.append(team_games_df)

# Concatenating all the dataframes for all teams into a single dataframe
all_teams_games_df = pd.concat(all_teams_games, ignore_index=True)

# Displaying the first few rows of the final dataframe
all_teams_games_df.head()

# Call the function 'get_player_shotchartdetail' with 'Bradley Beal' and '2023-24' as arguments
    .
# This function fetches the shot chart details for Bradley Beal for the 2023-24 season.
# The function returns two dataframes:
# 'player_shotchart_df' contains the shot chart details for Bradley Beal,
# 'league_avg' contains the league average shot chart details for the 2023-24 season.
player_shotchart_df, league_avg = get_player_shotchartdetail('Bradley Beal', '2023-24')

# Prints the first few rows of the player_shotchart_df dataframe
player_shotchart_df

# This line of code checks if there are any null values in the 'LOC_X' column of the '
    player_shotchart_df' DataFrame.
# 'isnull()' returns a DataFrame where each cell is either True or False depending on whether
    that cell's value is null.
# The first 'any()' reduces this DataFrame to a Series, where each element is True if any value
     in the corresponding column is True.
# The second 'any()' further reduces this Series to a single boolean value, which is True if
    any value in the Series is True.
# In other words, this line of code will return True if there are any null values in the 'LOC_X
    ' column, and False otherwise.
player_shotchart_df['LOC_X'].isnull().any().any()

# Prints the first few rows of the league_avg dataframe
league_avg

def draw_court(ax=None, color="blue", lw=1, shotzone=False, outer_lines=False):

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

    # Create the various parts of an NBA basketball court

    # Create the basketball hoop
    hoop = Circle((0, 0), radius=7.5, linewidth=lw, color=color, fill=False)

    # Create backboard
    backboard = Rectangle((-30, -12.5), 60, 0, linewidth=lw, color=color)

    # The paint
    # Create the outer box Of the paint, width=16ft, height=19ft
    outer_box = Rectangle((-80, -47.5), 160, 190, linewidth=lw, color=color,
                        fill=False)
    # Create the inner box of the paint, widt=12ft, height=19ft
    inner_box = Rectangle((-60, -47.5), 120, 190, linewidth=lw, color=color,
                        fill=False)
```

```
# 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 right side 3pt lines, it's 14ft long before it arcs
corner_three_a = Rectangle((-220, -47.5), 0, 140, linewidth=lw,
                           color=color)
# Create the right side 3pt lines, it's 14ft long before it arcs
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 Court
center_outer_arc = Arc((0, 422.5), 120, 120, theta1=180, theta2=0,
                       linewidth=lw, color=color)
center_inner_arc = Arc((0, 422.5), 40, 40, theta1=180, theta2=0,
                       linewidth=lw, color=color)

# Draw shotzone Lines
# Based on Advanced Zone Mode
if (shotzone == True):
    inner_circle = Circle((0, 0), radius=80, linewidth=lw, color='black', fill=False)
    outer_circle = Circle((0, 0), radius=160, linewidth=lw, color='black', fill=False)
    corner_three_a_x = Rectangle((-250, 92.5), 30, 0, linewidth=lw, color=color)
    corner_three_b_x = Rectangle((220, 92.5), 30, 0, linewidth=lw, color=color)

    # 60 degrees
    inner_line_1 = Rectangle((40, 69.28), 80, 0, 60, linewidth=lw, color=color)
    # 120 degrees
    inner_line_2 = Rectangle((-40, 69.28), 80, 0, 120, linewidth=lw, color=color)

    # Assume x distance is also 40 for the endpoint
    inner_line_3 = Rectangle((53.20, 150.89), 290, 0, 70.53, linewidth=lw, color=color)
    inner_line_4 = Rectangle((-53.20, 150.89), 290, 0, 109.47, linewidth=lw, color=color)

    # Assume y distance is also 92.5 for the endpoint
    inner_line_5 = Rectangle((130.54, 92.5), 80, 0, 35.32, linewidth=lw, color=color)
    inner_line_6 = Rectangle((-130.54, 92.5), 80, 0, 144.68, linewidth=lw, color=color)


    # List of the court elements to be plotted onto the axes
    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, inner_circle, outer_circle,
                      corner_three_a_x, corner_three_b_x,
                      inner_line_1, inner_line_2, inner_line_3, inner_line_4, inner_line_5,
                          inner_line_6]
else:
    # List of the court elements to be plotted onto the axes
    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:
        # Draw the half court line, baseline and side out bound lines
        outer_lines = Rectangle((-250, -47.5), 500, 470, linewidth=lw,
                                color=color, fill=False)
        court_elements.append(outer_lines)

    # Add the court elements onto the axes
    for element in court_elements:
        ax.add_patch(element)


    return ax

# Create a new figure and axes with a size of 12x12 inches
fig, ax = plt.subplots(figsize=(12, 12))

# Set the background color of the axes to black
ax.set_facecolor('black')

# Create a heatmap of the shot chart data using seaborn's kdeplot function
# The x and y coordinates are the locations of the shots
# The 'fill' parameter is set to True to fill in the areas of the heatmap
# The 'cmap' parameter is set to 'inferno' to use the inferno color map
# The 'ax' parameter is set to the previously created axes
# The 'bw_adjust' parameter is set to 1 to adjust the bandwidth of the kernel density estimate
sns.kdeplot(data=player_shotchart_df, x='LOC_X', y='LOC_Y', fill=True, cmap='inferno', ax=ax,
    bw_adjust=1)

# Add scatter points for individual shots on top of the heatmap
# The x and y coordinates are the locations of the shots
# The color of the points is set to white
# The size of the points is set to 2
# The alpha of the points is set to 0.2 to make them semi-transparent
ax.scatter(player_shotchart_df['LOC_X'], player_shotchart_df['LOC_Y'], color='white', s=2,
    alpha=0.2)

# Draw the basketball court on the axes
draw_court(ax)

# Set the x and y limits of the plot to match the dimensions of the basketball court
ax.set_xlim(-250, 250)
ax.set_ylim(-47.5, 422.5)

# Set the title of the plot to "Bradley Beal All Shots 23-24 Heat Map"
ax.set_title("Bradley Beal All Shots 23-24 Heat Map", fontsize=18)

# Display the plot
plt.show()

# Create a new figure and axes with a size of 12x12 inches
fig, ax = plt.subplots(figsize=(12, 12))

# Set the background color of the axes to black
ax.set_facecolor('black')

# Create a new DataFrame that only contains the made shots
# This is done by filtering the 'player_shotchart_df' DataFrame where the 'EVENT_TYPE' column
    is 'Made Shot'
made_shots_df = player_shotchart_df[player_shotchart_df['EVENT_TYPE'] == 'Made Shot']

# Create the heatmap for made shots using seaborn's kdeplot function
# The x and y coordinates are the locations of the made shots
# The 'fill' parameter is set to True to fill in the areas of the heatmap
# The 'cmap' parameter is set to 'inferno' to use the inferno color map
# The 'ax' parameter is set to the previously created axes
# The 'bw_adjust' parameter is set to 1 to adjust the bandwidth of the kernel density estimate
sns.kdeplot(data=made_shots_df, x='LOC_X', y='LOC_Y', fill=True, cmap='inferno', ax=ax,
    bw_adjust=1)

# Add white scatter points for individual shots on top of the heatmap
# The x and y coordinates are the locations of the shots
# The color of the points is set to white
# The size of the points is set to 2
# The alpha of the points is set to 0.2 to make them semi-transparent
ax.scatter(player_shotchart_df['LOC_X'], player_shotchart_df['LOC_Y'], color='white', s=2,
    alpha=0.2)

# Draw the basketball court on the axes
draw_court(ax)

# Set the x and y limits of the plot to match the dimensions of the basketball court
ax.set_xlim(-250, 250)
ax.set_ylim(-47.5, 422.5)

# Set the title of the plot to "Bradley Beal Made Shots 23-24 Heat Map"
ax.set_title("Bradley Beal Made Shots 23-24 Heat Map", fontsize=18)

# Display the plot
plt.show()

# Create a new figure and axes with a size of 12x12 inches
fig, ax = plt.subplots(figsize=(12, 12))

# Set the background color of the axes to black
ax.set_facecolor('black')

# Create a new DataFrame that only contains the missed shots
# This is done by filtering the 'player_shotchart_df' DataFrame where the 'EVENT_TYPE' column
    is 'Missed Shot'
made_shots_df = player_shotchart_df[player_shotchart_df['EVENT_TYPE'] == 'Missed Shot']

# Create the heatmap for missed shots using seaborn's kdeplot function
# The x and y coordinates are the locations of the missed shots
# The 'fill' parameter is set to True to fill in the areas of the heatmap
# The 'cmap' parameter is set to 'inferno' to use the inferno color map
# The 'ax' parameter is set to the previously created axes
# The 'bw_adjust' parameter is set to 1 to adjust the bandwidth of the kernel density estimate
sns.kdeplot(data=made_shots_df, x='LOC_X', y='LOC_Y', fill=True, cmap='inferno', ax=ax,
    bw_adjust=1)

# Add white scatter points for individual shots on top of the heatmap
# The x and y coordinates are the locations of the shots
# The color of the points is set to white
# The size of the points is set to 2
# The alpha of the points is set to 0.2 to make them semi-transparent
ax.scatter(player_shotchart_df['LOC_X'], player_shotchart_df['LOC_Y'], color='white', s=2,
    alpha=0.2)

# Draw the basketball court on the axes
draw_court(ax)

# Set the x and y limits of the plot to match the dimensions of the basketball court
ax.set_xlim(-250, 250)
ax.set_ylim(-47.5, 422.5)

# Set the title of the plot to "Bradley Beal Missed shots 23-24 Heat Map"
```

```
ax.set_title("Bradley Beal Missed shots 23-24 Heat Map", fontsize=18)

# Display the plot
plt.show()

# Define a function to adjust the size of hexbins based on their count and color based on
    another value
def sized_hexbin(ax, hc, hc2, cmap, norm):
    # Get the center positions of the hexbins
    offsets = hc.get_offsets()
    # Get the original path (hexagon shape) of the hexbins
    orgpath = hc.get_paths()[0]
    # Get the vertices of the original path
    verts = orgpath.vertices
    # Get the count values of the hexbins
    values1 = hc.get_array()
    # Get the color values of the hexbins
    values2 = hc2.get_array()
    # Get the maximum count value
    ma = values1.max()
    # Initialize an empty list to store the new hexbin patches
    patches = []

    # Loop over each hexbin
    for offset,val in zip(offsets,values1):
        # Skip hexbins with a count of 0
        if (int(val) == 0):
            continue
        # Create a list of non-zero count values
        filtered_list = list(filter(lambda num: num != 0, values1))
        # Adjust the size of the hexbin based on its count value
        if (percentileofscore(filtered_list, val) < 33.33):
            v1 = verts*0.3 + offset
        elif (percentileofscore(filtered_list, val) > 69.99):
            v1 = verts + offset
        else:
            v1 = verts*0.6 + offset
        # Create a new path for the hexbin
        path = Path(v1, orgpath.codes)
        # Create a new patch for the hexbin
        patch = PathPatch(path)
        # Add the new patch to the list
        patches.append(patch)

    # Create a collection of patches with the specified colormap and normalization
    pc = PatchCollection(patches, cmap=cmap, norm=norm)
    # Set the color values of the patches
    pc.set_array(values2)
    # Add the patch collection to the axes
    ax.add_collection(pc)
    # Remove the original hexbins
    hc.remove()
    hc2.remove()

# Define a function to create a hexbin map chart
def hexmap_chart(data, league_avg, title="", color="b",
             xlim=(-250, 250), ylim=(422.5, -47.5), line_color="white",
             court_color="#1a477b", court_lw=2, outer_lines=False,
             flip_court=False, gridsize=None,
             ax=None, despine=False, **kwargs):

    # Calculate the league average field goal percentage for each shot zone
```

```
LA = league_avg.loc[:,['SHOT_ZONE_AREA','SHOT_ZONE_RANGE', 'FGA', 'FGM']].groupby(['
    SHOT_ZONE_AREA', 'SHOT_ZONE_RANGE']).sum()
LA['FGP'] = 1.0*LA['FGM']/LA['FGA']
# Calculate the player's field goal percentage for each shot zone
player = data.groupby(['SHOT_ZONE_AREA','SHOT_ZONE_RANGE','SHOT_MADE_FLAG']).size().unstack(
    fill_value=0)
player['FGP'] = 1.0*player.loc[:,1]/player.sum(axis=1)
# Calculate the difference between the player's field goal percentage and the league average
player_vs_league = (player.loc[:,'FGP'] - LA.loc[:,'FGP'])*100
# Merge the difference data with the original data
data = pd.merge(data, player_vs_league, on=['SHOT_ZONE_AREA', 'SHOT_ZONE_RANGE'], how='right
    ')

# Create a new axes if none is provided
if ax is None:
   ax = plt.gca()
   ax.set_facecolor(court_color)

# Set the limits of the axes
if not flip_court:
   ax.set_xlim(xlim)
   ax.set_ylim(ylim)
else:
   ax.set_xlim(xlim[::-1])
   ax.set_ylim(ylim[::-1])

# Remove the axis labels
ax.tick_params(labelbottom="off", labelleft="off")
# Set the title of the plot
ax.set_title(title, fontsize=18)

# Get the x and y coordinates of the shots
x = data['LOC_X']
y = data['LOC_Y']

# Define a diverging colormap
colors = ['#30011E', '#550068', '#BB3754', '#FA8E63', '#FED788', '#FCFFA4']
cmap = ListedColormap(colors)
# Define the boundaries for the colormap
boundaries = [-np.inf, -9, -3, 0, 3, 9, np.inf]
norm = BoundaryNorm(boundaries, cmap.N, clip=True)

# Create two hexbin plots, one for the count values and one for the color values
hexbin = ax.hexbin(x, y, gridsize=gridsize, cmap=cmap, norm=norm, extent=[-275, 275, -50,
    425])
hexbin2 = ax.hexbin(x, y, C=data['FGP'], gridsize=gridsize, cmap=cmap, norm=norm, extent
    =[-275, 275, -50, 425])
# Adjust the size and color of the hexbins
sized_hexbin(ax, hexbin, hexbin2, cmap, norm)

# Create a legend for the colormap
labels = ["< 9% less", "9% to 3% less", "3% to 0% less", "3% to 0% more", "3% to 9% more",
    ">9% more"]
patches = [mpatches.Patch(color=c, label=l) for c, l in zip(colors, labels)]
ax.legend(handles=patches, bbox_to_anchor=(1.05, 1), loc='upper left', title='FG% vs. League
     Average (%)', title_fontsize='large', fontsize='large')

# Draw the basketball court on the axes
draw_court(ax, color='black', lw=court_lw, outer_lines=outer_lines)

# Set the line width and color of the axes spines
for spine in ax.spines:
   ax.spines[spine].set_lw(court_lw)
```

```
        ax.spines[spine].set_color(line_color)

    # Remove the axes spines if specified
    if despine:
        ax.spines["top"].set_visible(False)
        ax.spines["bottom"].set_visible(False)
        ax.spines["right"].set_visible(False)
        ax.spines["left"].set_visible(False)

    # Return the axes
    return ax

# Call the hexmap_chart function with the player's shot chart data, the league average data, a
    title, and a grid size
# This will create a hexbin map chart that shows the difference between Bradley Beal's field
    goal percentage and
# the league average for each shot zone in the 2023-24 season
hexmap_chart(player_shotchart_df, league_avg, title="Bradley Beal Hex Chart 2023-24", gridsize
    =30)

# Display the plot
plt.show()

# Filter the DataFrame 'all_teams_games_df' where the 'TEAM_ABBREVIATION' is 'PHX' (Phoenix
    Suns), and get the 'TEAM_ID'
suns_team_id = all_teams_games_df[all_teams_games_df['TEAM_ABBREVIATION'] == 'PHX']['TEAM_ID']

# Filter the DataFrame 'all_teams_games_df' where the 'TEAM_ABBREVIATION' is 'MIN' (Minnesota
    Timberwolves), and get the 'TEAM_ID'
twolves_team_id = all_teams_games_df[all_teams_games_df['TEAM_ABBREVIATION'] == 'MIN']['TEAM_ID
    ']

# Filter the DataFrame 'player_shotchart_df' where the 'PLAYER_NAME' is 'Bradley Beal', and get
     the 'PLAYER_ID'
bradley_beal_id = player_shotchart_df[player_shotchart_df['PLAYER_NAME'] == 'Bradley Beal']['
    PLAYER_ID'].values[0]

# Use the ShotChartDetail function from the shotchartdetail module to get shot chart data
# The parameters specify that we want data for Bradley Beal (player_id=bradley_beal_id) playing
     for the Phoenix Suns (team_id=suns_team_id)
# against the Minnesota Timberwolves (opponent_team_id=twolves_team_id) in the 2023-24 playoffs
# (season_nullable='2023-24', season_type_all_star='Playoffs')
# The context_measure_simple parameter is set to "FGA" to get data for field goal attempts
# The get_data_frames()[0] at the end gets the first DataFrame from the returned data
brad_wolves_suns = shotchartdetail.ShotChartDetail(team_id=suns_team_id, player_id=
    bradley_beal_id,
                                               opponent_team_id=twolves_team_id,
                                               season_type_all_star='Playoffs',
                                               season_nullable='2023-24',
                                               context_measure_simple="FGA").
                                                   get_data_frames()[0]

# Call the hexmap_chart function with the shot chart data, the league average data, a title,
    and a grid size
# This will create a hexbin map chart that shows the difference between Bradley Beal's field
    goal percentage
# and the league average for each shot zone in the 2023-24 playoffs against the Timberwolves
hexmap_chart(brad_wolves_suns, league_avg, title="Bradley Beal Hex Chart against Timberwolves
    2023-24 playoffs", gridsize=10)

# Display the plot
plt.show()
```

```
# Use the ShotChartDetail function from the shotchartdetail module to get shot chart data
# The parameters specify that we want data for the Phoenix Suns (team_id=1610612756) against
    the Minnesota Timberwolves
# (opponent_team_id=1610612750) in the 2023-24 playoffs (season_nullable='2023-24',
    season_type_all_star='Playoffs')
# The context_measure_simple parameter is set to "FGA" to get data for field goal attempts
# The get_data_frames()[0] at the end gets the first DataFrame from the returned data
wolves_suns = shotchartdetail.ShotChartDetail(team_id=1610612756, player_id=0,
                                                   opponent_team_id=1610612750,
                                                   season_type_all_star='Playoffs',
                                                   season_nullable='2023-24',
                                                   context_measure_simple="FGA").
                                                       get_data_frames()[0]

# Call the hexmap_chart function with the shot chart data, the league average data, a title,
    and a grid size
# This will create a hexbin map chart that shows the difference between the Suns' field goal
    percentage and the
# league average for each shot zone in the 2023-24 playoffs against the Timberwolves
hexmap_chart(wolves_suns, league_avg, title="Suns Hex Chart against Timberwolves 2023-24
    playoffs", gridsize=15)

# Display the plot
plt.show()

# Define a function to get game information by matchup
def get_game_info_by_matchup(matchup_value, games_df):
    # Filter the games DataFrame based on the provided matchup value
    filtered_df = games_df[games_df['MATCHUP'] == matchup_value]

    # Extract the GAME_ID and GAME_DATE columns from the filtered DataFrame
    game_info_df = filtered_df[['GAME_ID', 'GAME_DATE']]

    # Return the DataFrame with game information
    return game_info_df

# Assuming all_teams_games_df contains the DataFrame with all NBA games data

# Define the matchup value as "PHX @ MIN" (Phoenix Suns at Minnesota Timberwolves)
matchup_value = "PHX @ MIN"

# Call the get_game_info_by_matchup function with the matchup value and the games DataFrame
# This will return a DataFrame with the game ID and date for all games where the Suns played at
     the Timberwolves
game_info = get_game_info_by_matchup(matchup_value, all_teams_games_df)

# Print the DataFrame with game information
print(game_info)

# Import the HustleStatsBoxScore endpoint from the nba_api.stats.endpoints module
from nba_api.stats.endpoints import hustlestatsboxscore

# Define a function to fetch hustle data for a specific game
def fetch_hustle_data(game_id):
    try:
        # Call the HustleStatsBoxScore endpoint with the game ID to get hustle data
        # The get_data_frames() method returns the data as a list of DataFrames
        hustle_data = hustlestatsboxscore.HustleStatsBoxScore(game_id=game_id).get_data_frames()
        # Return the hustle data
        return hustle_data
    except Exception as e:
        # If an error occurs, print an error message and return None
        print(f"Failed to fetch boxscore data for game {game_id}: {e}")
        return None

# Define the game ID for the game between the Suns and the Timberwolves Game 1 in the playoffs
game_id = '0042300161'

# Call the fetch_hustle_data function with the game ID to get the hustle data
hustle_df = fetch_hustle_data(game_id)

# If the hustle data was successfully fetched
if hustle_df is not None:
    # Print a success message
    print("Boxscore data successfully fetched:")
    # Loop through each DataFrame in the hustle data
    for df_index, df in enumerate(hustle_df):
        # Print the index and the DataFrame
        print(f"DataFrame {df_index + 1}:")
        print(df)
        # Convert the DataFrame to a pandas DataFrame (if it's not already)
        df = pd.DataFrame(df)
        # Print the columns of the DataFrame
        print("Columns:")
        print(df.columns)
# If the hustle data was not fetched due to an error
else:
    # Print an error message
    print("Boxscore data was not fetched due to an error.")

# This DataFrame contains team-level hustle stats
team_hustle = hustle_df[2]

# Display the DataFrame
team_hustle

import matplotlib.pyplot as plt

# Define a function to get team data by team name from the hustle data
def get_team_data_by_name(team_name, hustle_df):
    # Extract data for the specified team name from hustle_df
    team_data = hustle_df[2]
    team_data = team_data[team_data['TEAM_NAME'] == team_name]
    return team_data

# Define a function to compare impact metrics between two teams
def compare_impact_metrics(team1_data, team2_data, team1_name, team2_name):
    # Define impact metrics to compare
    metrics = ['PTS', 'CONTESTED_SHOTS', 'DEFLECTIONS', 'CHARGES_DRAWN',
               'SCREEN_ASSISTS', 'OFF_LOOSE_BALLS_RECOVERED', 'DEF_LOOSE_BALLS_RECOVERED',
               'LOOSE_BALLS_RECOVERED', 'OFF_BOXOUTS', 'DEF_BOXOUTS']

    # Extract values for team 1 and team 2
    team1_metrics = team1_data[metrics].iloc[0].values
    team2_metrics = team2_data[metrics].iloc[0].values

    # Create bar plot
    fig, ax = plt.subplots(figsize=(12, 8))
    index = range(len(metrics))
    bar_width = 0.35

    # Create bars for team 1 and team 2
    team1_bars = ax.bar(index, team1_metrics, bar_width, label=team1_name, color='b')
    team2_bars = ax.bar([i + bar_width for i in index], team2_metrics, bar_width, label=
        team2_name, color='r')
    # Add labels and title
    ax.set_xlabel('Metrics')
    ax.set_ylabel('Values')
    ax.set_title('Comparison of Hustle Metrics between ' + team1_name + ' and ' + team2_name + '
         game 1 23/24 playoffs')
    ax.set_xticks([i + bar_width / 2 for i in index])
    ax.set_xticklabels(metrics, rotation=45, ha='right')
    ax.legend()

    # Add grid
    ax.grid(True, which='both', linestyle='--', linewidth=0.5)

    # Add annotations to bars
    for i in index:
        ax.annotate(f'{team1_metrics[i]}', xy=(i, team1_metrics[i]), xytext=(0, 3),
                    textcoords="offset points", ha='center', va='bottom', color='black')
        ax.annotate(f'{team2_metrics[i]}', xy=(i + bar_width, team2_metrics[i]), xytext=(0, 3),
                    textcoords="offset points", ha='center', va='bottom', color='black')

    # Adjust layout and display the plot
    plt.tight_layout()
    plt.show()

# Assuming hustle_df contains the box-score data for both teams
# Get data for the Timberwolves and the Suns
team1_data = get_team_data_by_name('Timberwolves', hustle_df)
team2_data = get_team_data_by_name('Suns', hustle_df)

# Define team names
team1_name = 'Timberwolves'
team2_name = 'Suns'

# Compare impact metrics between the Timberwolves and the Suns
compare_impact_metrics(team1_data, team2_data, team1_name, team2_name)

from nba_api.stats.static import teams
from nba_api.stats.endpoints import leaguegamefinder, hustlestatsboxscore
import pandas as pd
from functools import wraps
import time

def retry(ExceptionToCheck, tries=4, delay=3, backoff=2, logger=None):
    def deco_retry(f):
        @wraps(f)
        def f_retry(*args, **kwargs):
            mtries, mdelay = tries, delay
            while mtries > 1:
                try:
                    return f(*args, **kwargs)
                except ExceptionToCheck as e:
                    msg = f"{str(e)}, Retrying in {mdelay} seconds..."
                    if logger:
                        logger.warning(msg)
                    else:
                        print(msg)
                    time.sleep(mdelay)
                    mtries -= 1
                    mdelay *= backoff
            return f(*args, **kwargs)
        return f_retry
    return deco_retry

# Retry decorator for API calls
```

```
retry_api_call = retry(Exception)

nba_teams = teams.get_teams()

total_teams = len(nba_teams)
progress_step = 100 / total_teams

# Only get data for the 2023-2024 season
season_id = "2023-24"
all_teams_hustle_data = []

for idx, team in enumerate(nba_teams):
    team_id = team['id']
    team_abbreviation = team['abbreviation']

    team_hustle_data = []

    @retry_api_call
    def fetch_team_games(team_id, season_id):
        gamefinder = leaguegamefinder.LeagueGameFinder(team_id_nullable=team_id, season_nullable
            =season_id)
        games = gamefinder.get_data_frames()[0]
        return games

    try:
        games = fetch_team_games(team_id, season_id)
    except Exception as e:
        print(f"Failed to fetch games for team {team_abbreviation}: {e}")
        continue

    for game_id in games['GAME_ID']:
        try:
            hustle_data = hustlestatsboxscore.HustleStatsBoxScore(game_id=game_id).
                get_data_frames()
            if hustle_data:
                team_hustle_data.extend(hustle_data)
        except Exception as e:
            print(f"Failed to fetch hustle data for game {game_id}: {e}")

    if team_hustle_data:
        team_hustle_df = pd.concat(team_hustle_data, ignore_index=True)
        team_hustle_df['TEAM_ABBREVIATION'] = team_abbreviation
        all_teams_hustle_data.append(team_hustle_df)

    # Print progress
    progress = (idx + 1) * progress_step
    print(f"Progress: {progress:.2f}%")

# Store all the data in a single DataFrame
all_teams_hustle_df = pd.concat(all_teams_hustle_data, ignore_index=True)

# Save the DataFrame 'all_teams_hustle_df' to a CSV file named 'all_teams_hustle_df.csv'.
# The 'index=False' argument means that the DataFrame's index will not be saved in the file.
all_teams_hustle_df.to_csv('all_teams_hustle_df.csv', index=False)

# Read the CSV file 'all_teams_hustle_df.csv' into a DataFrame named 'hustle_2023_24'.
hustle_2023_24 = pd.read_csv('all_teams_hustle_df.csv')

# Drop the rows in the DataFrame 'hustle_2023_24' where the 'TEAM_NAME' is missing (NaN).
# The 'subset' argument specifies that only NaN values in the 'TEAM_NAME' column should be
    considered.
hustle_2023_24 = hustle_2023_24.dropna(subset=['TEAM_NAME'])
```

```
# Display the DataFrame 'hustle_2023_24'.
hustle_2023_24

# Drop the duplicate rows in the DataFrame 'hustle_2023_24' where the 'GAME_ID' and 'TEAM_NAME'
     are the same.
# The 'subset' argument specifies that only duplicates in the 'GAME_ID' and 'TEAM_NAME' columns
     should be considered.
hustle_2023_24 = hustle_2023_24.drop_duplicates(subset=['GAME_ID', 'TEAM_NAME'])

# Display the DataFrame 'hustle_2023_24'.
hustle_2023_24

# First, sort the DataFrame by 'GAME_ID' and 'PTS'
hustle_2023_24 = hustle_2023_24.sort_values(by=['GAME_ID', 'PTS'])

# Then, group by 'GAME_ID' and assign 0 to the team with fewer points and 1 to the team with
    more points
hustle_2023_24['W/L'] = hustle_2023_24.groupby('GAME_ID').cumcount()

# Reverse the 'W/L' values (so that 1 represents a win and 0 represents a loss)
hustle_2023_24['W/L'] = hustle_2023_24['W/L'].map({0: 1, 1: 0})

hustle_2023_24

# Define a list of column names that we want to calculate differences for
columns_to_diff = ['CONTESTED_SHOTS', 'CONTESTED_SHOTS_2PT', 'CONTESTED_SHOTS_3PT',
                 'DEFLECTIONS', 'CHARGES_DRAWN', 'SCREEN_ASSISTS', 'SCREEN_AST_PTS',
                 'OFF_LOOSE_BALLS_RECOVERED', 'DEF_LOOSE_BALLS_RECOVERED',
                 'LOOSE_BALLS_RECOVERED', 'OFF_BOXOUTS', 'DEF_BOXOUTS',
                 'BOX_OUT_PLAYER_TEAM_REBS', 'BOX_OUT_PLAYER_REBS', 'BOX_OUTS']

# For each column in the list
for column in columns_to_diff:
    # Create a new column name by appending '_DIFF' to the original column name
    diff_column = column + '_DIFF'

    # Calculate the difference between each row and the previous row within each 'GAME_ID' group
    # and store the results in the new column
    hustle_2023_24[diff_column] = hustle_2023_24.groupby('GAME_ID')[column].diff()

    # Fill any missing values in the new column with the negative of the next row's value
    hustle_2023_24[diff_column] = hustle_2023_24[diff_column].fillna(-hustle_2023_24[diff_column
        ].shift(-1))

# Display the DataFrame 'hustle_2023_24'
hustle_2023_24

# Drop the non-numeric columns from the DataFrame 'hustle_2023_24'.
# The 'columns' argument specifies the names of the columns to be dropped.
hustle_2023_24 = hustle_2023_24.drop(columns=['GAME_ID', 'HUSTLE_STATUS', 'TEAM_ID', '
    TEAM_ABBREVIATION', 'PLAYER_ID', 'PLAYER_NAME', 'START_POSITION', 'COMMENT', 'MINUTES', '
    PTS', 'TEAM_NAME'])

# Display the DataFrame 'hustle_2023_24'.
hustle_2023_24

# Drop the 'TEAM_CITY' column from the DataFrame 'hustle_2023_24'.
# The 'columns' argument specifies the name of the column to be dropped.
hustle_2023_24 = hustle_2023_24.drop(columns=['TEAM_CITY'])

# Display the DataFrame 'hustle_2023_24'.
hustle_2023_24
```

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import linregress

# Perform a linear regression for each stat with 'W/L' as the independent variable
# Store the results in a dictionary where the keys are the stat names and the values are the
    regression results
linreg_results = {column: linregress(hustle_2023_24['W/L'], hustle_2023_24[column]) for column
    in hustle_2023_24.columns if column != 'W/L'}

# Convert the dictionary of regression results into a DataFrame
# Transpose the DataFrame so that the stat names are the index and the regression results are
    the columns
linreg_df = pd.DataFrame(linreg_results).T
# Rename the columns of the DataFrame to reflect the regression results
linreg_df.columns = ['Slope', 'Intercept', 'R-value', 'P-value', 'Std Error']

# Create a new figure for the bar plot
plt.figure(figsize=(10, 8))
# Create a horizontal bar plot of the slopes for each stat
bars = plt.barh(linreg_df.index, linreg_df['Slope'])

# For each bar in the plot
for bar in bars:
    # Get the width of the bar (which is the slope value)
    width = bar.get_width()
    # Add a text label to the bar with the slope value
    # The label is positioned at the end of the bar and is centered vertically
    # The label is aligned to the right if the slope is negative and to the left if the slope is
        positive
    plt.text(width, bar.get_y() + bar.get_height()/2, f'{width:.2f}',
            va='center', ha='right' if width < 0 else 'left')

# Add a label to the x-axis
plt.xlabel('Slope')
# Add a title to the plot
plt.title('Slope of Linear Regression for Each Stat')
# Display the plot
plt.show()
```

2 The Data

The statistics used to create our data visualizations come from the National Basketball Association Application Programming Interface repository, or NBA API repository for short. The NBA API repository is a free tool that utilizes Python code in order to download and display statistics from the NBA official statistics website, nba.com/stats. The NBA statistics themselves are retrieved during the live games utilizing cameras installed on the catwalks in every NBA arena. Second Spectrum software is then used to track the movement of every player during every second of the game, locating where each individual is 25 times per second. Second Spectrum, by Genius Sports, is a company that works with large national corporations like the NBA, NFL, and Premier League to provide spectators and the corporations themselves with the newest most advanced breakdowns of recent games using advanced machine learning and computer vision techniques in order to locate every player on a given plane with extreme accuracy. Second Spectrum is the official optical tracking provider of both the NBA and Premier League. If desired by spectators, Second Spectrum is able to deliver automated pre-game, post-game, and player reports to any given inbox in the form of a PDF. Second Spectrum is able to calculate things like player field goal percentage, passes, and more with very high precision. The NBA then takes these statistics and utilizes them on its website for easier consumption.

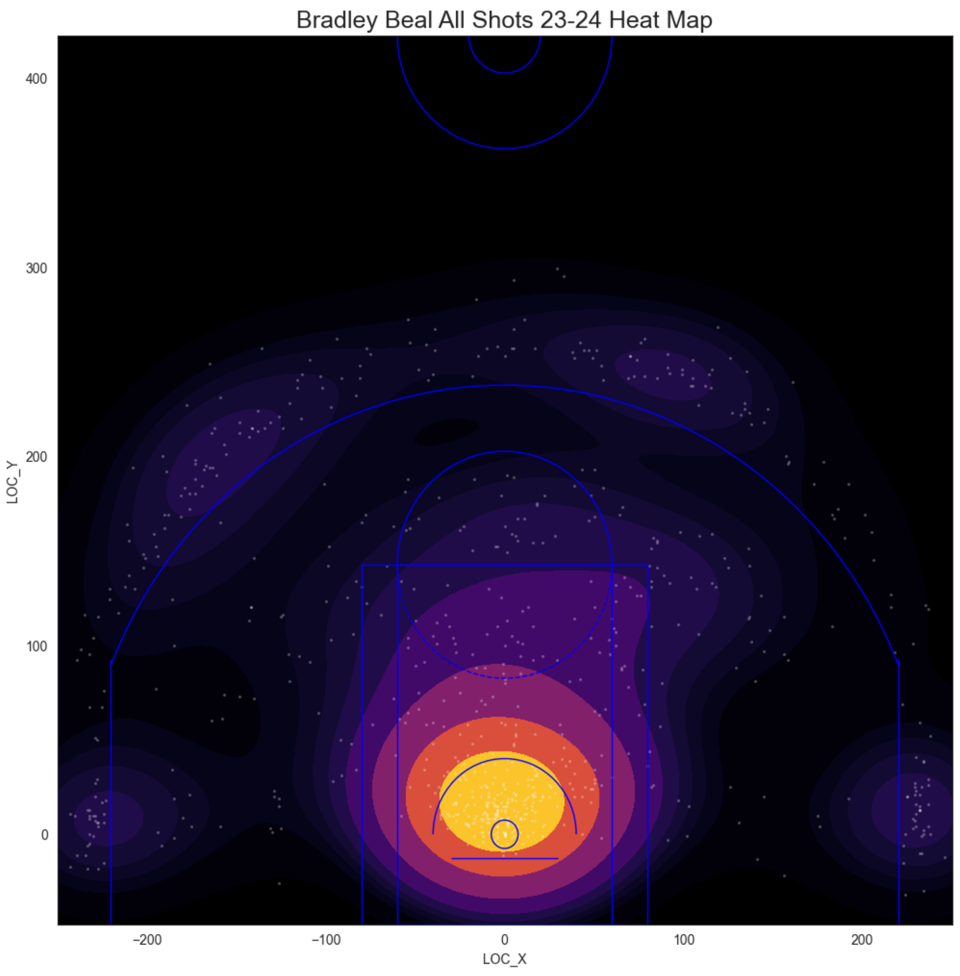
The API repository code, the method of retrieving all game data from the NBA website, was originally created by Swar Patel and Randy Forbes and was originally released on September 16, 2018. Although these two gentlemen were the source of the code, there are almost 30 contributors in total. Not only does the code contain over a hundred endpoints of NBA statistics from all teams and players and games from many seasons, but the API is constantly being updated and edited in order to provide users with the most up-to-date information from the NBA, as well as creating a package that is easy to use and retrieve data from. In the heat maps, hex charts, and bar graphs that we were able to create with our original code, the data from the API repository was utilized. By specifying the type of data we required for different applications (like shots made vs shots attempted in order to calculate shooting percentage), we were able to call specific data from individual players or teams from any given season with ease, and in turn, learn more about the defensive disruption that different teams were able to implement upon said players and teams.

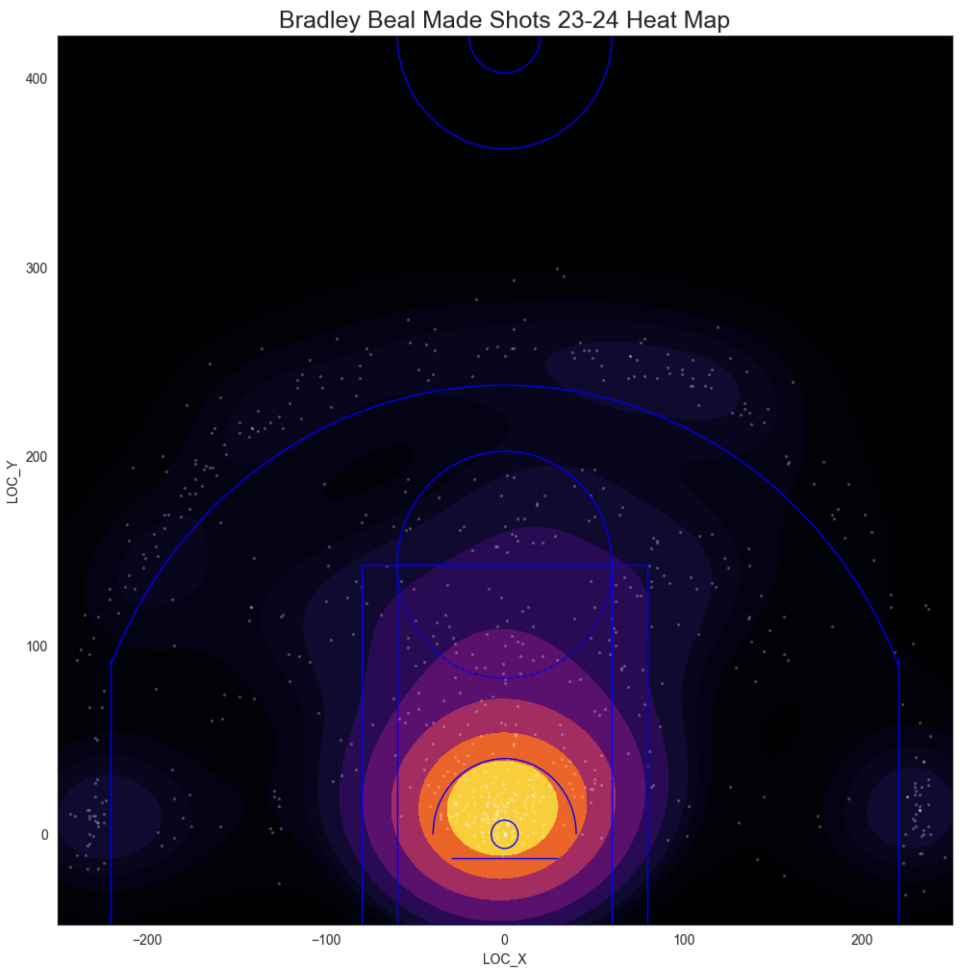
3 Mathematical Insights - Heat Maps

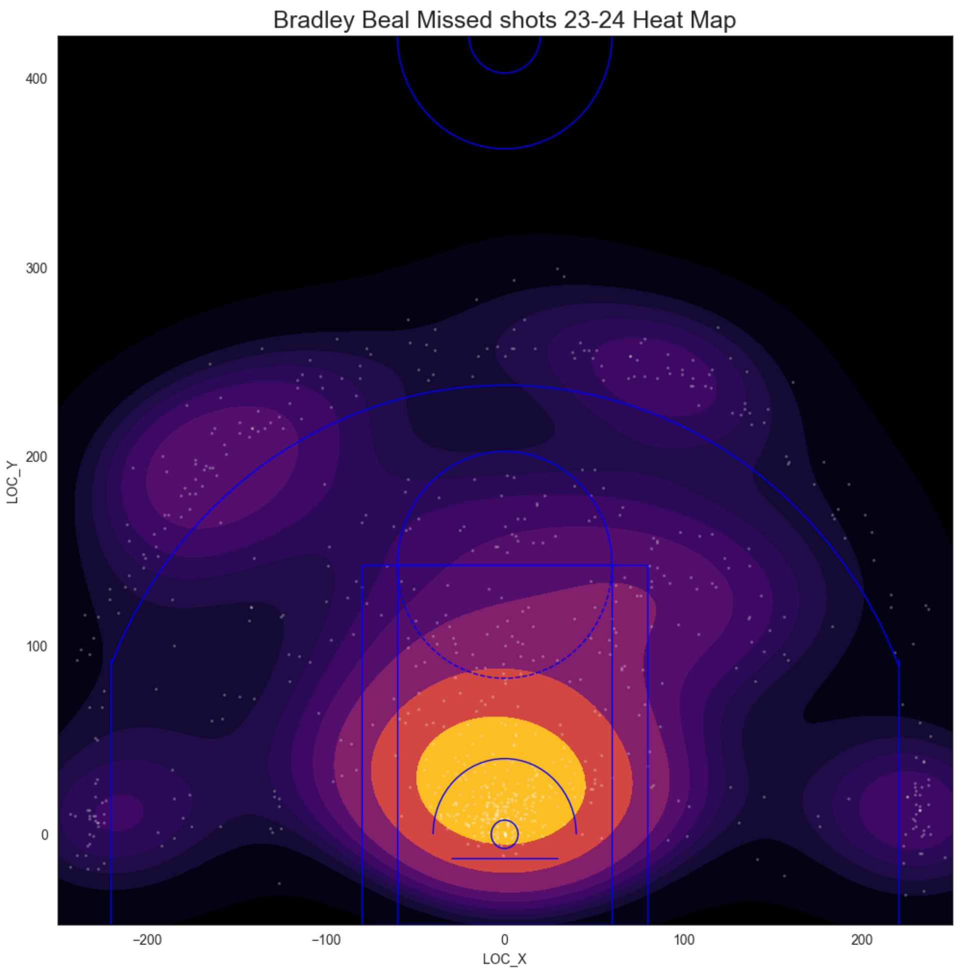
In order to mathematically gauge the defensive disruption of a given team on a given player or team, we utilized a few different techniques. To begin, we retrieved the exact locations of each shot taken from the shot chart data. In the NBA API, each x and y location is scaled by 0.1 feet, i.e., a shot taken from $(-80, 50)$ was taken 8 feet to the left of the basket and 5 feet above when looking directly down at the basket facing the half-court line. This scale was also used to plot our basketball court, on which the shot locations are plotted. The court is scaled up by 10, and the origin is set at the middle of the basket. Therefore, since an NBA-regulated court is 50 ft wide, the x values span from -250 to 250 (-25 to 25 ft from the basket), and since the basket is 4.75 feet from the baseline, the y values span from -47.5 to 422.5, the half-court is 47 ft long, hence $-47.5 + 422.5 = 470$.

In order to create the heat maps, we solely utilized the data representing the shots taken, made, and missed from each location of the court. Once we retrieved the data, we then plotted each point on the court we mapped above and applied Seaborn's (a data visualization library based on matplotlib) `kdeplot` function (kernel density plot). This function plots a small 'kernel' at each data point, and once all the points are plotted, combines the kernels into a probability density plot based on the bandwidth value (smaller values, like the one we chose, 1, provide more detail as each kernel is smaller and the probability areas are more detailed in shape). Since there are 8 different colors, spanning from bright orange to black, representing high to low probability density respectively, each bin represents a 12.5% difference in the probability of appearing in one of the specific colors, i.e., an $87.5\% - 99.9999\%$ chance of a shot taken in the bright orange kernel, and a $0\% - 12\%$ chance of a shot taken in the black kernel.

By examining the heat maps of Bradley Beal (shown below), we were able to conclude that there is an 87.5% chance that any given made shot was taken from within the paint (around where the bright orange kernel is located) and $50\% - 62.5\%$ $(100 - 4 \times 12.5\%)$ chance that any missed shot was taken around the key or along the 3-pt line (where the 4th color from bright orange is located on the missed shots heatmap).


Bradley Beal All Shots 23-24 Heat Map
400
300
200
100
0
LOC_Y
−200
−100
0
100
200
LOC_X


Bradley Beal Made Shots 23-24 Heat Map
400
300
200
100
0
LOC_Y
−200
−100
0
100
200
LOC_X



4 Mathematical Insights - HexCharts

The HexCharts had the purpose of comparing player and team field goal percentages (FG%) against both league and opposing team FG%. In order to accomplish this, we first had to calculate player, team, and league FG% themselves. In order to do this, we took a simple percentage calculation of made vs attempted shots, i.e. $\frac{\text{made shots}}{\text{attempted shots}}$. This equation results in the percentage of shots that a given player or team as well as the league would make overall. After these values were calculated, all that was left was to find a way to compare them to each other. To do this, we first calculated the difference in FG% between the two metrics of interest (if we wanted to calculate the difference between Bradley Beal's FG% and the league FG% overall, we would find the difference, and multiply by 100, to make the distinction between each hex bin easier to determine). We then assigned the color of each hex bin to a range of difference in FG% (0-3% more or less is represented by 2 different colors, 3-9% more or less by two more colors, and ¿9% more or less by two more).

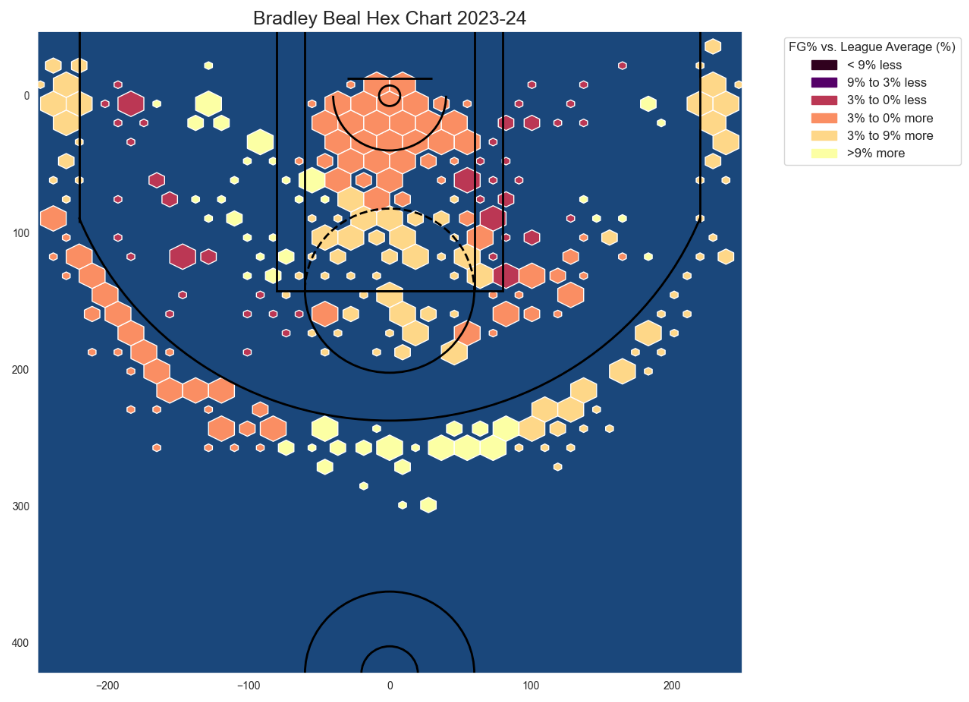
The size of each hex bin is determined by the *sized_hexbin* function. First, the function gets the count values of the hexbins, then for each hexbin, it calculates the percentile rank of its count value among all count values between 0 and 100 indicating the percentage of count values that are less than or equal to the count value of the current hexbin. Based on this percentile rank, it adjusts the size of the hexbin. If the percentile rank is less than 33.33, it scales the size of the hexbin down by a factor of 0.3. If the percentile rank is greater than 69.99, it keeps the size of the hexbin as it is. Otherwise, it scales the size of the hexbin down by a factor of 0.6. Finally, it creates a new patch for the hexbin with the adjusted size and adds it to the list of patches. In short, the hexbins with a higher count value will be larger and hexbins with a lower count value will be smaller. The size is adjusted based on the percentile rank of the count value among all count values.

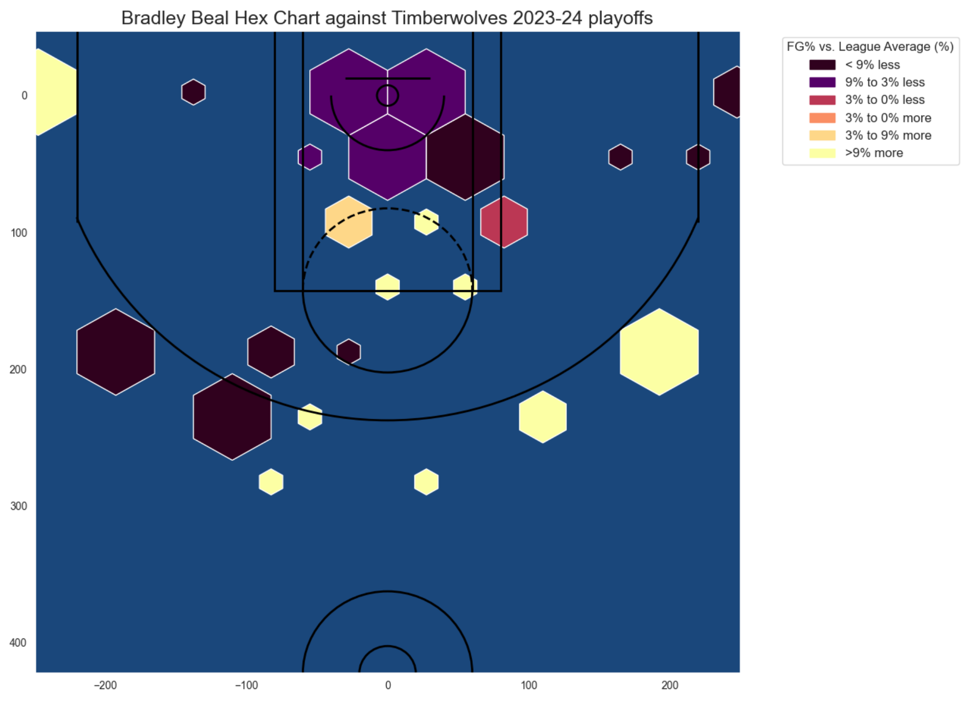
Examining the first HexChart below, we are able to determine that Bradley Beal has around the same FG% if not slightly higher (0-3% higher) than the league overall closer to the basket, and around the same FG% if not slightly less (0-3% less) than the league overall along the right side of the key. The

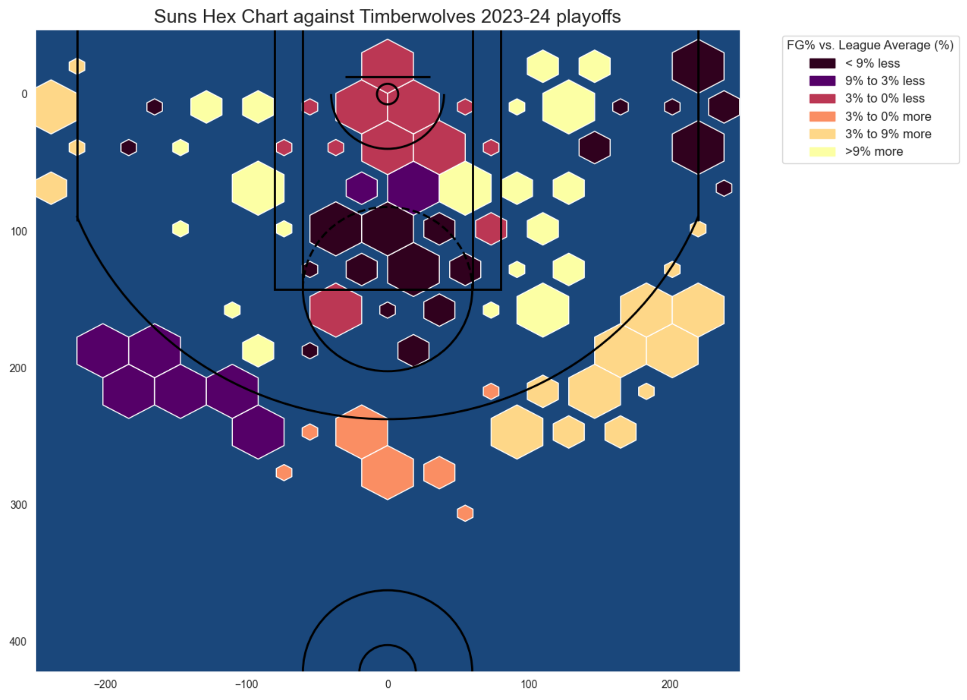
HexChart makes it simple to determine this, as the darker orange and red hexbins correspond to these FG% differences.

Looking at the second HexChart below, it is clear that the Timberwolves were effectively able to defensively disrupt Bradley Beal's game in the 2023-24 playoffs, as all of his FG% within the paint were within the dark purple (3-9% less) or maroon (>9% less) hex bins when compared to the league average overall, extremely different than the hexbins when looking at the first chart (Bradley Beal vs the League FG% overall in the 2023-24 season). This isn't too surprising considering the Timberwolves had 7 foot 1 2023-24 defensive player of the year Rudy Gobert playing defense in the paint and Bradley Beal is a much smaller two-guard.

Looking at the final HexChart below, it is safe to say that the Timberwolves were easily able to defensively disrupt not only Beal but all of the Phoenix Suns in the 2023-24 playoffs. Not only are the majority of the hexbins within the paint red, purple, or maroon, indicating a 0-3%, 3-9%, and >9% negative difference when comparing to overall league FG%, but the bins are also very large, indicating a very high concentration of shots in those areas. The Suns were swept in the series and didn't score over 100 points until game 3!

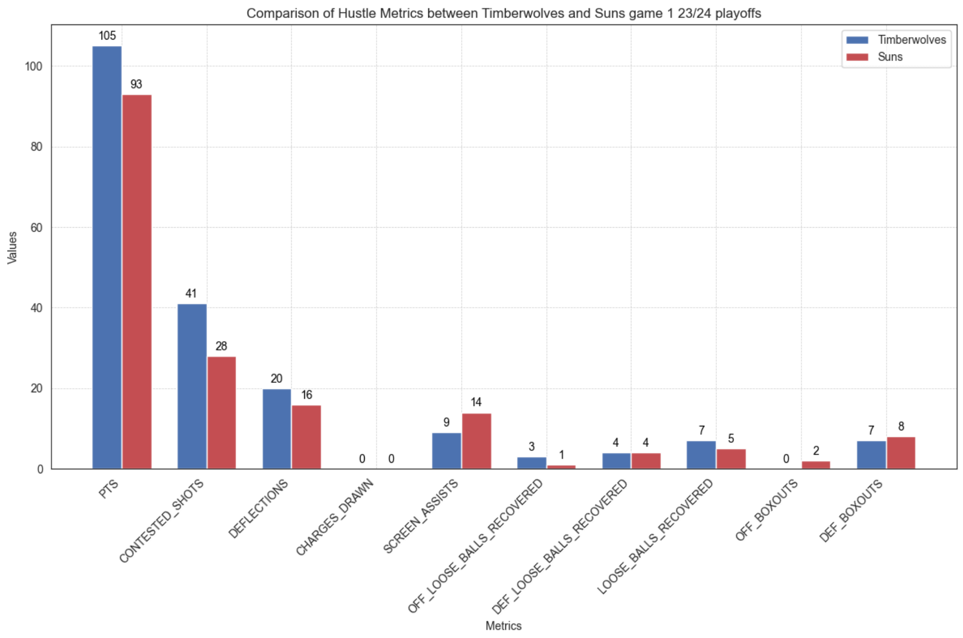






5 Mathematical Insights - Bar Graphs

While the HexCharts are good for comparing individual player or team FG% in reference to overall league FG%, there are a lot of stats that are not shown by the HexCharts. When looking at the defensive disruption of one team on another (in this case, The Suns vs The Timberwolves), a lot of important data can be found within the 'hustle_df', a data frame containing many important defensive metrics such as deflections, contested shots, and loose balls recovered. When comparing The Suns' and Timberwolves' performance against each other in the 2023-24 season, we decided that a bar graph showcasing each team's hustle stats, with each statistic type along the x-axis and quantity on the y-axis, was appropriate. When looking at the bar graph below, it is clear to see that not only did The Timberwolves score a significant win over The Suns as far as points are concerned, but The Timberwolves were also able to complete more deflections and recover more loose balls than The Suns. Although The Timberwolves did have to shoot significantly more contested shots than The Suns, meaning that The Suns were actively defending more attempted shots by The Timberwolves than vice-versa, The Timberwolves still won the game by a significant margin while having contested significantly less of The Sun's shots. This means that overall, The Timberwolves' defense was much more effective than The Suns', and The Timberwolves were able to defensively disrupt The Suns much more effectively than The Suns were able to defensively disrupt The Timberwolves in the 2023-24 season.



6 Mathematical Insights - Hustle Stats Linear Regression

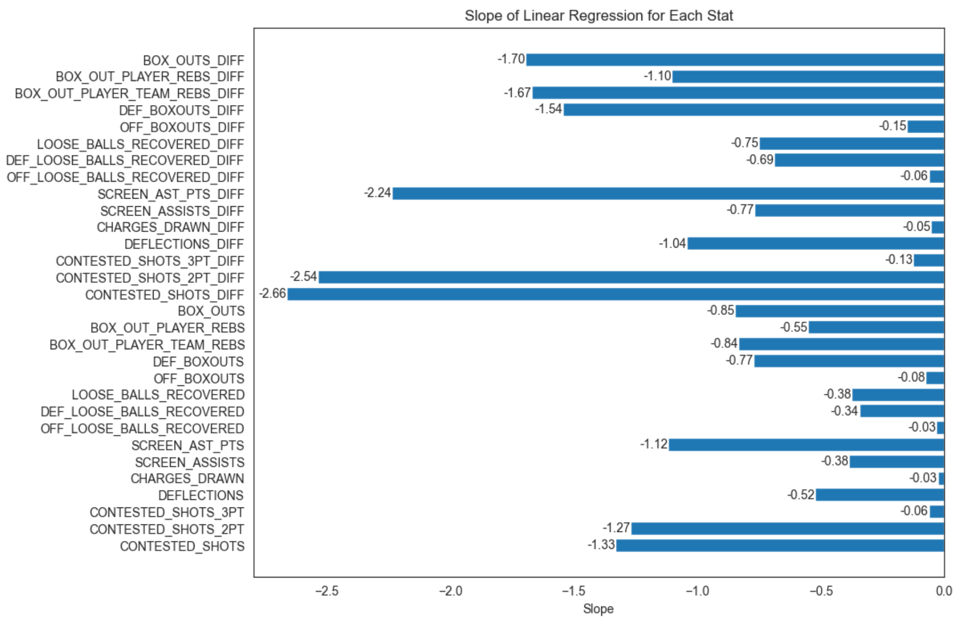
Linear regression is a statistical method that allows us to study the relationship between two continuous variables. In this case, we're looking at the relationship between the 'W/L' column (win or loss) and each of the other statistical categories. The basic formula for a linear regression is $y = mx + b$, where:

- y is the dependent variable (the statistical category in this case),
- x is the independent variable ('W/L' in this case),
- m is the slope of the line, and
- b is the y-intercept.

The slope m and y-intercept b are determined such that the line best fits the data according to a certain criterion. In the case of ordinary least squares (the method used by `linregress`), the criterion is to minimize the sum of the squared residuals. A residual for a data point is the difference between the observed value of y and the value of y predicted by the regression line. The slope is the change in y for a one-unit change in x. In this context, it represents the change in the statistical category for a change from a loss to a win. The intercept is the expected value of y when x is 0. In this context, it represents the expected value of the statistical category for a loss.

The r-value (or correlation coefficient) measures the strength and direction of the relationship between x and y. It ranges from -1 to 1, with -1 indicating a strong negative relationship, 1 indicating a strong positive relationship, and 0 indicating no relationship. The p-value is used in hypothesis testing to help you determine the significance of your results. In general, a p-value less than 0.05 is taken to mean the results are statistically significant. The standard error measures the accuracy of the slope's estimate. A lower standard error means the slope is more accurate.

The bar plot created by the code visualizes the slopes of the regression lines for each statistical category. The slope values are also added as text labels on the bars. This allows you to see at a glance which statistical categories have a stronger relationship with the 'W/L' column. The lower absolute values for the Difference stats suggest that these stats have a weaker relationship with the 'W/L' value compared to the non-difference stats. However, the fact that all the Difference stats have negative slopes indicates a consistent trend: winning teams tend to have lower Differences in hustle stats with the other team. This could suggest that these stats are still important for understanding team performance, even if their impact is less direct or smaller in magnitude compared to other stats.



Bibliography

- https://pypi.org/project/nba_api/#history
- https://www.statsperform.com/#:˜:text=Stats%20Perform%20is%20the%20world,win%20audiences%2C%20custo
- https://www.nba.com/stats/help/faq
- https://www.secondspectrum.com/ourwork/teams-leagues/
- https://github.com/swar/nba_api