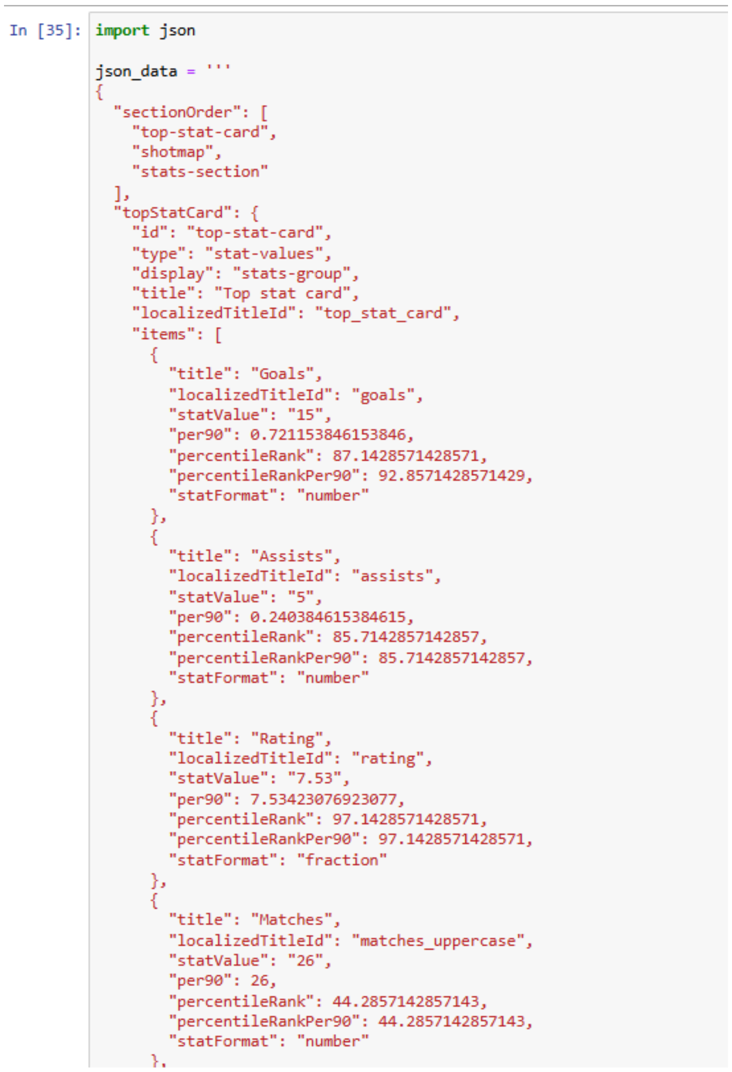
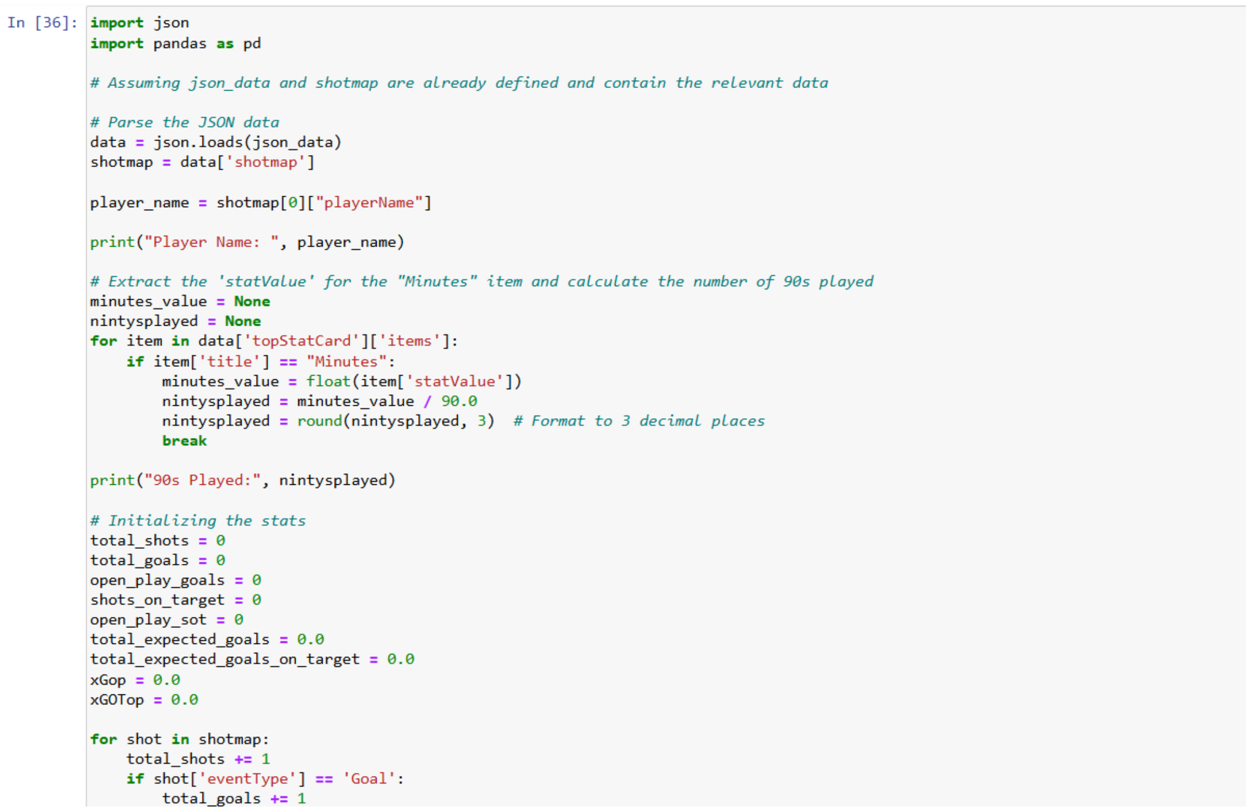
Automating Shot Quality Analysis Through Web Scraping

This project builds upon prior research on shot quality, aiming to streamline the shot map collection process through web scraping. By automating data collection, the project seeks to enhance efficiency and potentially pave the way for the development of a comprehensive website. This would serve as the culmination of nearly three years of dedicated research on the subject.

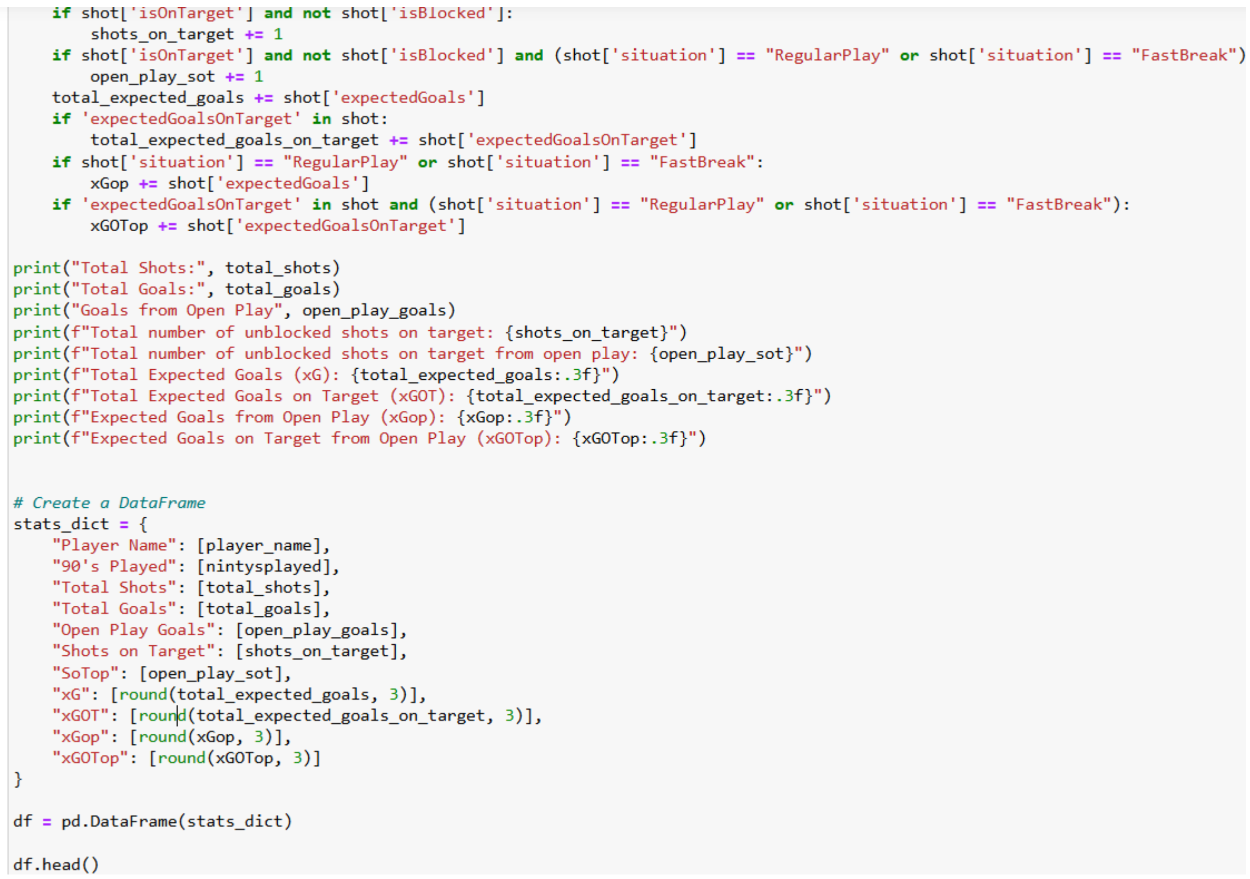


```
In [35]: import json

         json_data = '''
         {
           "sectionOrder": [
             "top-stat-card",
             "shotmap",
             "stats-section"
           ],
           "topStatCard": {
             "id": "top-stat-card",
             "type": "stat-values",
             "display": "stats-group",
             "title": "Top stat card",
             "localizedTitleId": "top_stat_card",
             "items": [
               {
                 "title": "Goals",
                 "localizedTitleId": "goals",
                 "statValue": "15",
                 "per90": 0.721153846153846,
                 "percentileRank": 87.1428571428571,
                 "percentileRankPer90": 92.8571428571429,
                 "statFormat": "number"
               },
               {
                 "title": "Assists",
                 "localizedTitleId": "assists",
                 "statValue": "5",
                 "per90": 0.240384615384615,
                 "percentileRank": 85.7142857142857,
                 "percentileRankPer90": 85.7142857142857,
                 "statFormat": "number"
               },
               {
                 "title": "Rating",
                 "localizedTitleId": "rating",
                 "statValue": "7.53",
                 "per90": 7.53423076923077,
                 "percentileRank": 97.1428571428571,
                 "percentileRankPer90": 97.1428571428571,
                 "statFormat": "fraction"
               },
               {
                 "title": "Matches",
                 "localizedTitleId": "matches_uppercase",
                 "statValue": "26",
                 "per90": 26,
                 "percentileRank": 44.2857142857143,
                 "percentileRankPer90": 44.2857142857143,
                 "statFormat": "number"
               },
```

By carefully utilizing the inspect element tool on www.fotmob.com and navigating to each player's shot map, then searching for "playerstats" to access the raw page data in a readable format, we can extract detailed information on every shot, including its description and location. By copying and pasting this raw JSON data into our IDE, we can efficiently extract shot coordinates without the need to manually locate shot maps as done previously—a process that was both tedious and prone to errors.

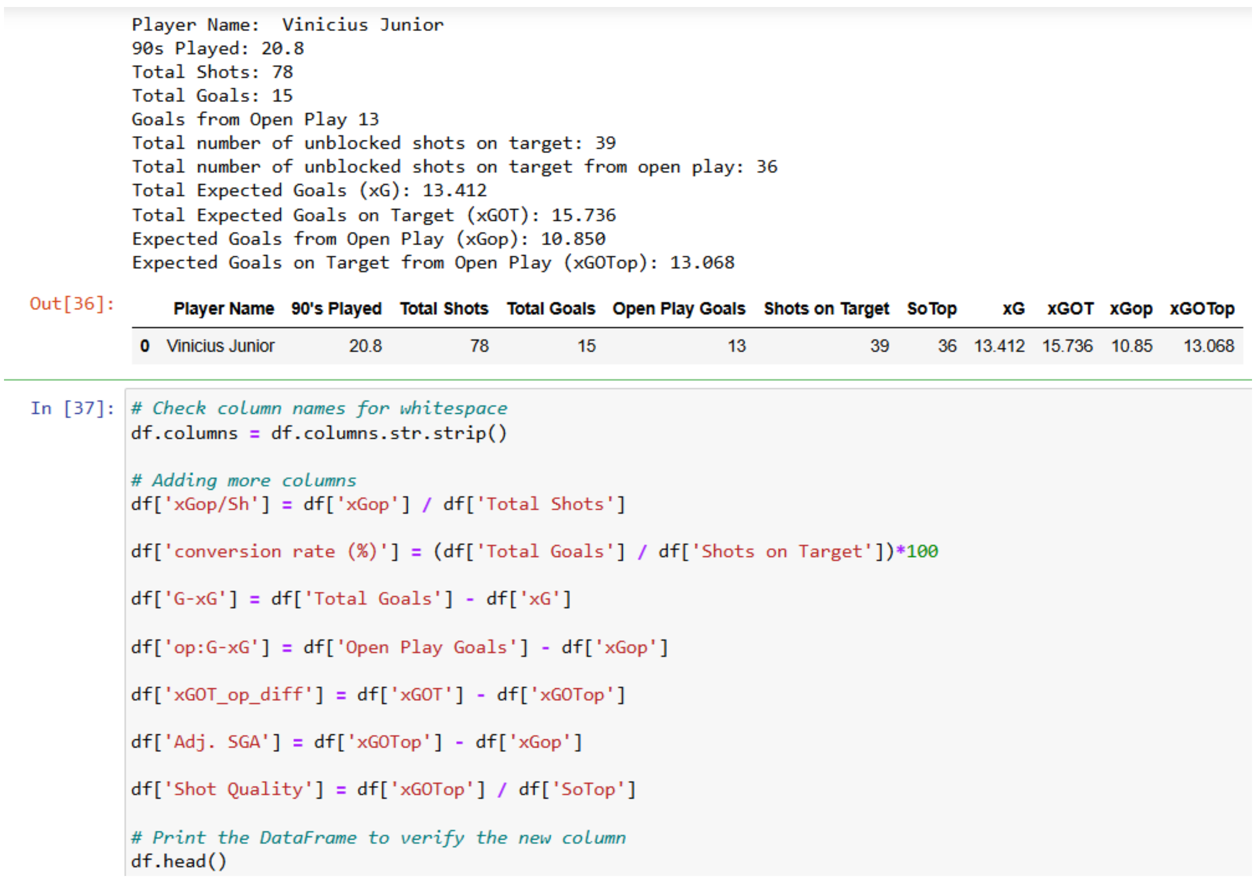


```
In [36]: import json
         import pandas as pd

         # Assuming json_data and shotmap are already defined and contain the relevant data

         # Parse the JSON data
         data = json.loads(json_data)
         shotmap = data['shotmap']

         player_name = shotmap[0]["playerName"]

         print("Player Name: ", player_name)

         # Extract the 'statValue' for the "Minutes" item and calculate the number of 90s played
         minutes_value = None
         nintysplayed = None
         for item in data['topStatCard']['items']:
             if item['title'] == "Minutes":
                 minutes_value = float(item['statValue'])
                 nintysplayed = minutes_value / 90.0
                 nintysplayed = round(nintysplayed, 3)  # Format to 3 decimal places
                 break

         print("90s Played:", nintysplayed)

         # Initializing the stats
         total_shots = 0
         total_goals = 0
         open_play_goals = 0
         shots_on_target = 0
         open_play_sot = 0
         total_expected_goals = 0.0
         total_expected_goals_on_target = 0.0
         xGop = 0.0
         xGOTop = 0.0

         for shot in shotmap:
             total_shots += 1
             if shot['eventType'] == 'Goal':
                 total_goals += 1
```



```python
    if shot['isOnTarget'] and not shot['isBlocked']:
        shots_on_target += 1
    if shot['isOnTarget'] and not shot['isBlocked'] and (shot['situation'] == "RegularPlay" or shot['situation'] == "FastBreak")
        open_play_sot += 1
    total_expected_goals += shot['expectedGoals']
    if 'expectedGoalsOnTarget' in shot:
        total_expected_goals_on_target += shot['expectedGoalsOnTarget']
    if shot['situation'] == "RegularPlay" or shot['situation'] == "FastBreak":
        xGop += shot['expectedGoals']
    if 'expectedGoalsOnTarget' in shot and (shot['situation'] == "RegularPlay" or shot['situation'] == "FastBreak"):
        xGOTop += shot['expectedGoalsOnTarget']

print("Total Shots:", total_shots)
print("Total Goals:", total_goals)
print("Goals from Open Play", open_play_goals)
print(f"Total number of unblocked shots on target: {shots_on_target}")
print(f"Total number of unblocked shots on target from open play: {open_play_sot}")
print(f"Total Expected Goals (xG): {total_expected_goals:.3f}")
print(f"Total Expected Goals on Target (xGOT): {total_expected_goals_on_target:.3f}")
print(f"Expected Goals from Open Play (xGop): {xGop:.3f}")
print(f"Expected Goals on Target from Open Play (xGOTop): {xGOTop:.3f}")


# Create a DataFrame
stats_dict = {
    "Player Name": [player_name],
    "90's Played": [nintysplayed],
    "Total Shots": [total_shots],
    "Total Goals": [total_goals],
    "Open Play Goals": [open_play_goals],
    "Shots on Target": [shots_on_target],
    "SoTop": [open_play_sot],
    "xG": [round(total_expected_goals, 3)],
    "xGOT": [round(total_expected_goals_on_target, 3)],
    "xGop": [round(xGop, 3)],
    "xGOTop": [round(xGOTop, 3)]
}

df = pd.DataFrame(stats_dict)

df.head()
```

This code processes raw JSON data containing a player's shot map statistics to calculate various performance metrics. It extracts the player's name, total minutes played (converting this to "90s Played"), and other key stats such as total shots, goals, open play goals, shots on target, and expected goals (xG) values. The metrics are further broken down to distinguish between open play and all play scenarios. Finally, the calculated statistics are organized into a structured DataFrame for easy analysis and visualization. This automated approach streamlines the analysis of player performance, replacing manual data entry with efficient, accurate extraction and computation of certain stats such as xGOT from open play which is used to derive the shot quality formula.



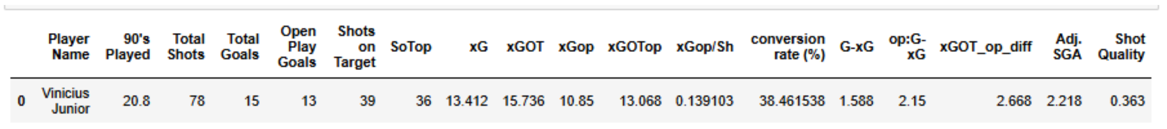
```
Player Name:  Vinicius Junior
90s Played: 20.8
Total Shots: 78
Total Goals: 15
Goals from Open Play 13
Total number of unblocked shots on target: 39
Total number of unblocked shots on target from open play: 36
Total Expected Goals (xG): 13.412
Total Expected Goals on Target (xGOT): 15.736
Expected Goals from Open Play (xGop): 10.850
Expected Goals on Target from Open Play (xGOTop): 13.068
```

Out[36]:

	Player Name	90's Played	Total Shots	Total Goals	Open Play Goals	Shots on Target	SoTop	xG	xGOT	xGop	xGOTop
0	Vinicius Junior	20.8	78	15	13	39	36	13.412	15.736	10.85	13.068

In [37]:

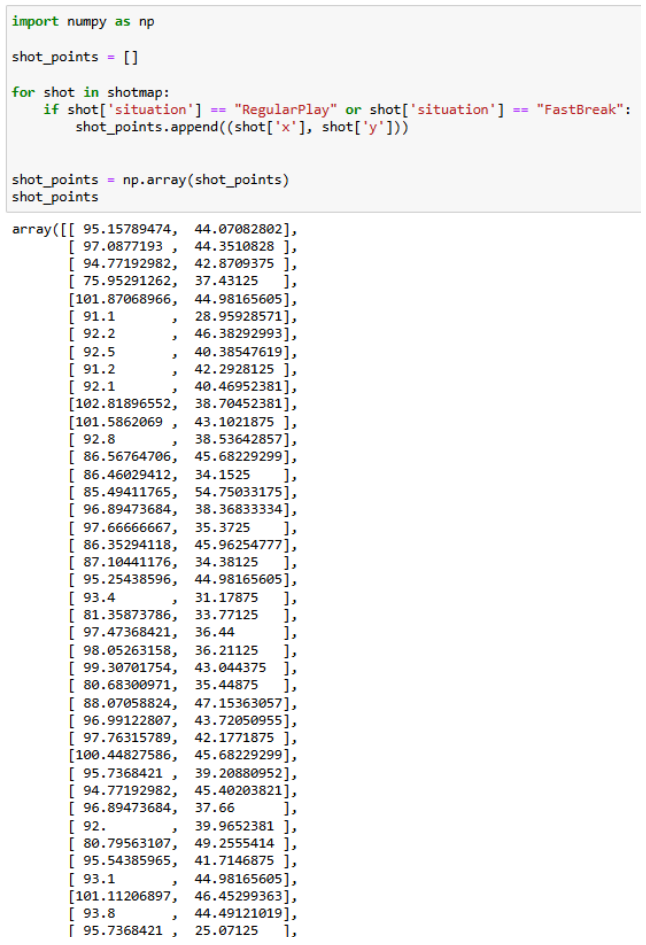
```python
# Check column names for whitespace
df.columns = df.columns.str.strip()

# Adding more columns
df['xGop/Sh'] = df['xGop'] / df['Total Shots']

df['conversion rate (%)'] = (df['Total Goals'] / df['Shots on Target'])*100

df['G-xG'] = df['Total Goals'] - df['xG']

df['op:G-xG'] = df['Open Play Goals'] - df['xGop']

df['xGOT_op_diff'] = df['xGOT'] - df['xGOTop']

df['Adj. SGA'] = df['xGOTop'] - df['xGop']

df['Shot Quality'] = df['xGOTop'] / df['SoTop']

# Print the DataFrame to verify the new column
df.head()
```



Out[37]:

	Player Name	90's Played	Total Shots	Total Goals	Open Play Goals	Shots on Target	SoTop	xG	xGOT	xGop	xGOTop	xGop/Sh	conversion rate (%)	G-xG	op:G-xG	xGOT_op_diff	Adj. SGA	Shot Quality
0	Vinicius Junior	20.8	78	15	13	39	36	13.412	15.736	10.85	13.068	0.139103	38.461538	1.588	2.15	2.668	2.218	0.363

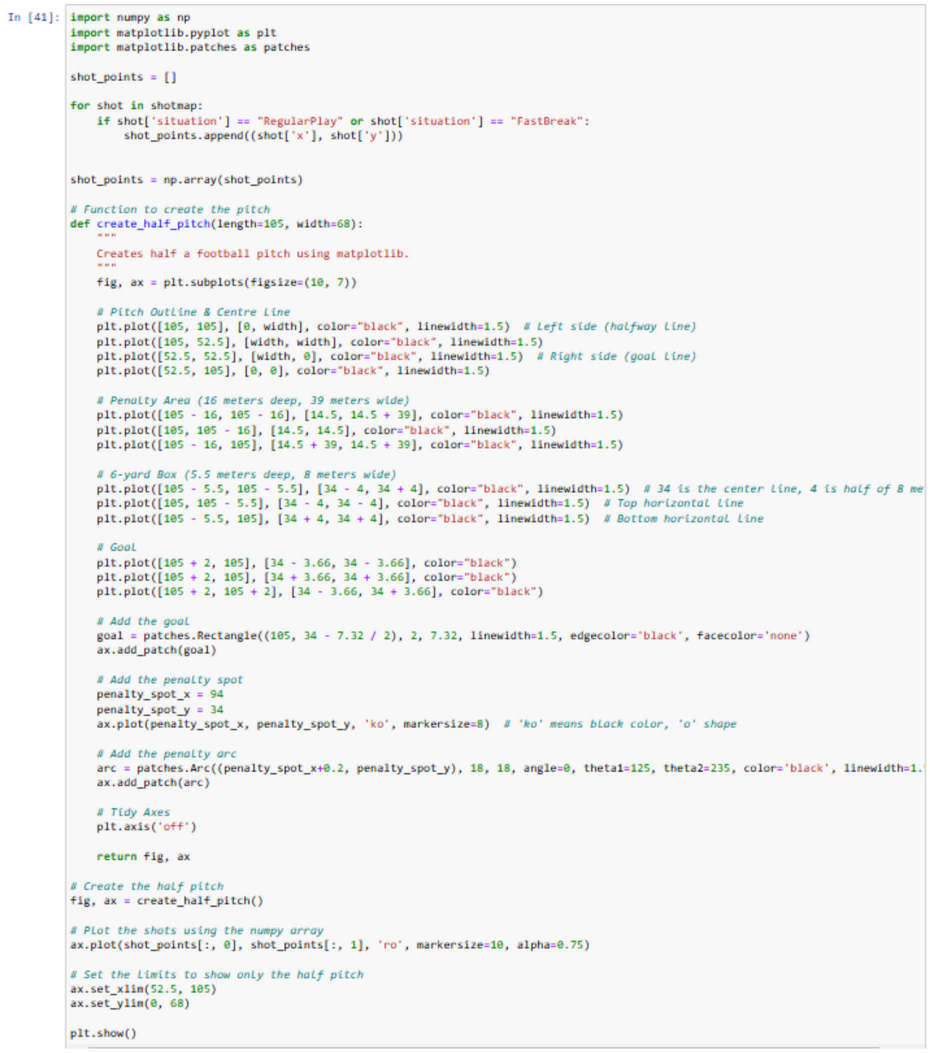
The code computes several advanced metrics to deepen the analysis of a player's shooting performance. It calculates **Expected Goals per Shot (xGop/Sh)** to measure the average quality of open play shots, providing insight into how likely each attempt is to result in a goal. The **Conversion Rate (%)** evaluates how efficiently a player converts shots on target into goals, highlighting finishing ability. The **Goal Difference from Expected Goals (G-xG)** shows whether a player is outperforming or underperforming relative to expected goals. Similarly, the **Open Play Goals Difference (op-Gop-xG)** focuses on goal efficiency specifically in open play scenarios. The **Difference between xGOT and xGOTop (xGOT_op_diff)** quantifies the impact of set-pieces and non-open play on shot placement quality. **Adjusted Shooting Goals Added (Adj. SGA)** reflects the player's added value from shot placement in open play, while **Shot Quality** provides a ratio of expected goals on target to shots on target in open play, offering a measure of shot placement efficiency. These metrics collectively enable a more comprehensive evaluation of a player's shooting effectiveness and decision-making in various game situations.



```python
import numpy as np

shot_points = []

for shot in shotmap:
    if shot['situation'] == "RegularPlay" or shot['situation'] == "FastBreak":
        shot_points.append((shot['x'], shot['y']))


shot_points = np.array(shot_points)
shot_points
```

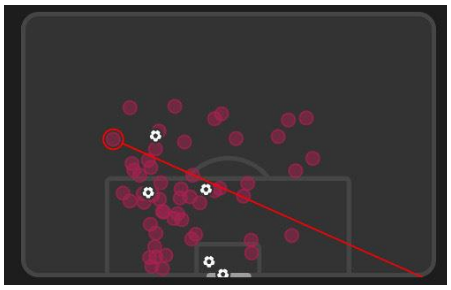
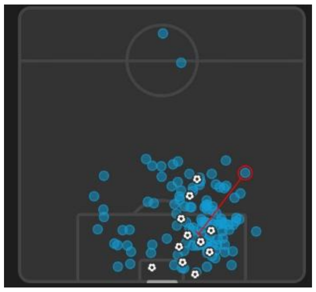
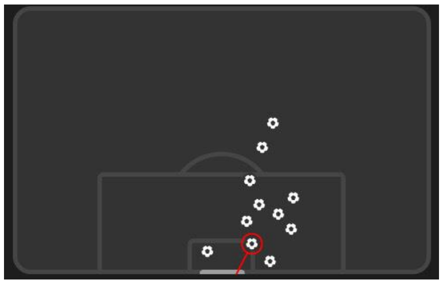
```
array([[ 95.15789474,  44.07082802],
       [ 97.0877193 ,  44.3510828 ],
       [ 94.77192982,  42.8709375 ],
       [ 75.95291262,  37.43125   ],
       [101.87068966,  44.98165605],
       [ 91.1       ,  28.95928571],
       [ 92.2       ,  46.38292993],
       [ 92.5       ,  40.38547619],
       [ 91.2       ,  42.2928125 ],
       [ 92.1       ,  40.46952381],
       [102.81896552,  38.70452381],
       [101.5862069 ,  43.1021875 ],
       [ 92.8       ,  38.53642857],
       [ 86.56764706,  45.68229299],
       [ 86.46029412,  34.1525    ],
       [ 85.49411765,  54.75033175],
       [ 96.89473684,  38.36833334],
       [ 97.66666667,  35.3725    ],
       [ 86.35294118,  45.96254777],
       [ 87.10441176,  34.38125   ],
       [ 95.25438596,  44.98165605],
       [ 93.4       ,  31.17875   ],
       [ 81.35873786,  33.77125   ],
       [ 97.47368421,  36.44      ],
       [ 98.05263158,  36.21125   ],
       [ 99.30701754,  43.044375  ],
       [ 80.68300971,  35.44875   ],
       [ 88.07058824,  47.15363057],
       [ 96.99122807,  43.72050955],
       [ 97.76315789,  42.1771875 ],
       [100.44827586,  45.68229299],
       [ 95.7368421 ,  39.20880952],
       [ 94.77192982,  45.40203821],
       [ 96.89473684,  37.66      ],
       [ 92.        ,  39.9652381 ],
       [ 80.79563107,  49.2555414 ],
       [ 95.54385965,  41.7146875 ],
       [ 93.1       ,  44.98165605],
       [101.11206897,  46.45299363],
       [ 93.8       ,  44.49121019],
       [ 95.7368421 ,  25.07125   ],
```

This code extracts all the shot coordinates from open play situations, specifically filtering for shots taken during "RegularPlay" or "FastBreak" scenarios. It iterates through the shotmap data, checks the situation attribute of each shot, and appends the shot's x and y coordinates to the shot_points list if it matches the criteria. Finally, it converts the list of shot points into a NumPy array for efficient numerical operations and further analysis. This step is crucial for visualizing and analyzing shot locations on the pitch, providing insights into a player’s shooting patterns during open play.

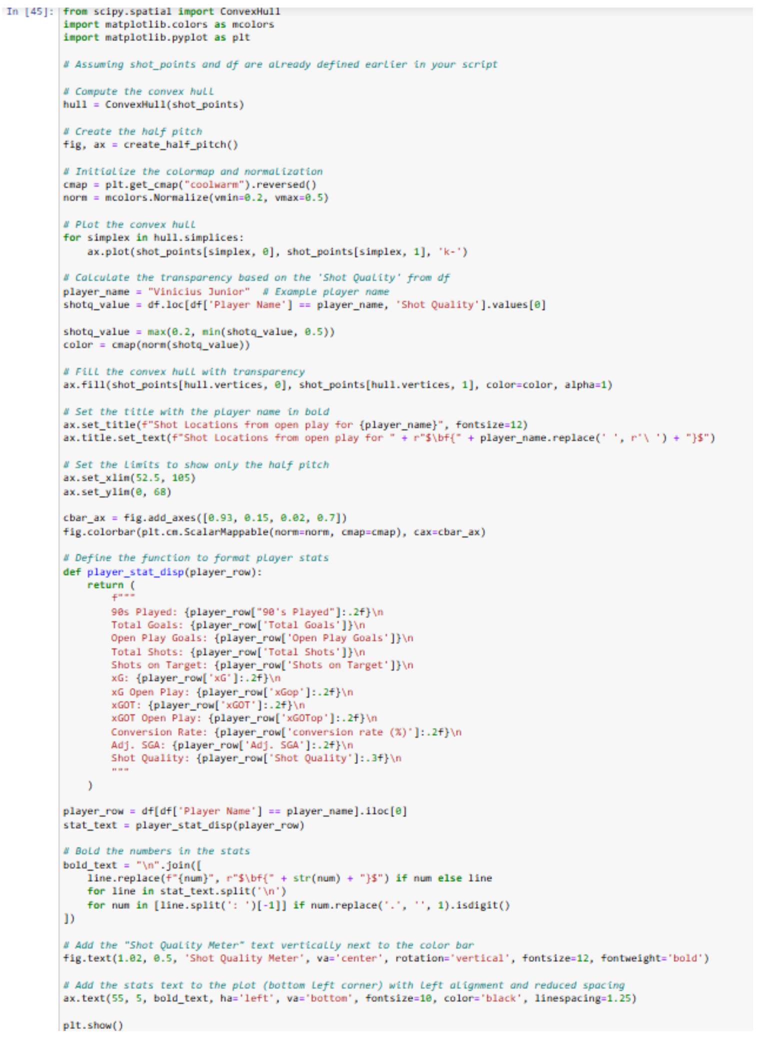


```
In [41]: import numpy as np
         import matplotlib.pyplot as plt
         import matplotlib.patches as patches

         shot_points = []

         for shot in shotmap:
             if shot['situation'] == "RegularPlay" or shot['situation'] == "FastBreak":
                 shot_points.append((shot['x'], shot['y']))


         shot_points = np.array(shot_points)

         # Function to create the pitch
         def create_half_pitch(length=105, width=68):
             """
             Creates half a football pitch using matplotlib.
             """
             fig, ax = plt.subplots(figsize=(10, 7))

             # Pitch Outline & Centre Line
             plt.plot([105, 105], [0, width], color="black", linewidth=1.5)  # Left side (halfway line)
             plt.plot([105, 52.5], [width, width], color="black", linewidth=1.5)
             plt.plot([52.5, 52.5], [width, 0], color="black", linewidth=1.5)  # Right side (goal line)
             plt.plot([52.5, 105], [0, 0], color="black", linewidth=1.5)

             # Penalty Area (16 meters deep, 39 meters wide)
             plt.plot([105 - 16, 105 - 16], [14.5, 14.5 + 39], color="black", linewidth=1.5)
             plt.plot([105, 105 - 16], [14.5, 14.5], color="black", linewidth=1.5)
             plt.plot([105 - 16, 105], [14.5 + 39, 14.5 + 39], color="black", linewidth=1.5)

             # 6-yard Box (5.5 meters deep, 8 meters wide)
             plt.plot([105 - 5.5, 105 - 5.5], [34 - 4, 34 + 4], color="black", linewidth=1.5)  # 34 is the center line, 4 is half of 8 me
             plt.plot([105, 105 - 5.5], [34 - 4, 34 - 4], color="black", linewidth=1.5)  # Top horizontal line
             plt.plot([105 - 5.5, 105], [34 + 4, 34 + 4], color="black", linewidth=1.5)  # Bottom horizontal line

             # Goal
             plt.plot([105 + 2, 105], [34 - 3.66, 34 - 3.66], color="black")
             plt.plot([105 + 2, 105], [34 + 3.66, 34 + 3.66], color="black")
             plt.plot([105 + 2, 105 + 2], [34 - 3.66, 34 + 3.66], color="black")

             # Add the goal
             goal = patches.Rectangle((105, 34 - 7.32 / 2), 2, 7.32, linewidth=1.5, edgecolor='black', facecolor='none')
             ax.add_patch(goal)

             # Add the penalty spot
             penalty_spot_x = 94
             penalty_spot_y = 34
             ax.plot(penalty_spot_x, penalty_spot_y, 'ko', markersize=8)  # 'ko' means black color, 'o' shape

             # Add the penalty arc
             arc = patches.Arc((penalty_spot_x+0.2, penalty_spot_y), 18, 18, angle=0, theta1=125, theta2=235, color='black', linewidth=1.
             ax.add_patch(arc)

             # Tidy Axes
             plt.axis('off')

             return fig, ax

         # Create the half pitch
         fig, ax = create_half_pitch()

         # Plot the shots using the numpy array
         ax.plot(shot_points[:, 0], shot_points[:, 1], 'ro', markersize=10, alpha=0.75)

         # Set the limits to show only the half pitch
         ax.set_xlim(52.5, 105)
         ax.set_ylim(0, 68)

         plt.show()
```

This code visualizes the shot locations from open play on a half-pitch football field. It first filters and stores the shot coordinates (x, y) from open play situations in a NumPy array. Using Matplotlib, it then defines a function create_half_pitch to draw a half-pitch with key features like the penalty area, 6-yard box, goal, and penalty spot. After creating the pitch, it plots the filtered shot points as red circles ('ro') on the pitch, providing a clear and accurate visual representation of where shots were taken. This helps in analyzing shot distribution and shooting patterns in open play.

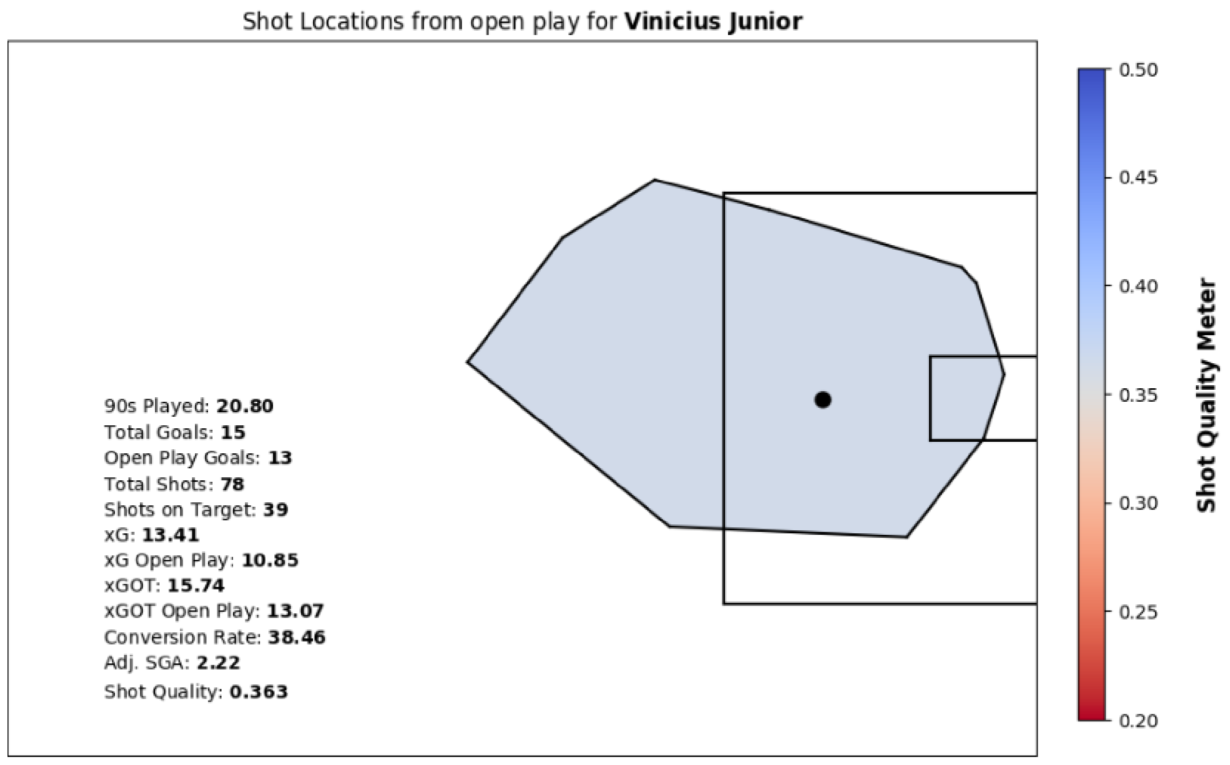
Problem Solving: I had to manually adjust the scaling of my pitch since the one used on Fotmob differs from the pitch dimensions I was previously working with. Fotmob uses unit measurements in meters, so I had to manually analyze the x/y values of specific shots. This process allowed me to fine-tune the scaling and align my pitch dimensions accurately.



```
In [45]: from scipy.spatial import ConvexHull
         import matplotlib.colors as mcolors
         import matplotlib.pyplot as plt

         # Assuming shot_points and df are already defined earlier in your script

         # Compute the convex hull
         hull = ConvexHull(shot_points)

         # Create the half pitch
         fig, ax = create_half_pitch()

         # Initialize the colormap and normalization
         cmap = plt.get_cmap("coolwarm").reversed()
         norm = mcolors.Normalize(vmin=0.2, vmax=0.5)

         # Plot the convex hull
         for simplex in hull.simplices:
             ax.plot(shot_points[simplex, 0], shot_points[simplex, 1], 'k-')

         # Calculate the transparency based on the 'Shot Quality' from df
         player_name = "Vinicius Junior"  # Example player name
         shotq_value = df.loc[df['Player Name'] == player_name, 'Shot Quality'].values[0]

         shotq_value = max(0.2, min(shotq_value, 0.5))
         color = cmap(norm(shotq_value))

         # Fill the convex hull with transparency
         ax.fill(shot_points[hull.vertices, 0], shot_points[hull.vertices, 1], color=color, alpha=1)

         # Set the title with the player name in bold
         ax.set_title(f"Shot Locations from open play for {player_name}", fontsize=12)
         ax.title.set_text(f"Shot Locations from open play for " + r"$\bf{" + player_name.replace(' ', r'\ ') + "}$")

         # Set the limits to show only the half pitch
         ax.set_xlim(52.5, 105)
         ax.set_ylim(0, 68)

         cbar_ax = fig.add_axes([0.93, 0.15, 0.02, 0.7])
         fig.colorbar(plt.cm.ScalarMappable(norm=norm, cmap=cmap), cax=cbar_ax)

         # Define the function to format player stats
         def player_stat_disp(player_row):
             return (
                 f"""
                 90s Played: {player_row["90's Played"]:.2f}\n
                 Total Goals: {player_row['Total Goals']}\n
                 Open Play Goals: {player_row['Open Play Goals']}\n
                 Total Shots: {player_row['Total Shots']}\n
                 Shots on Target: {player_row['Shots on Target']}\n
                 xG: {player_row['xG']:.2f}\n
                 xG Open Play: {player_row['xGop']:.2f}\n
                 xGOT: {player_row['xGOT']:.2f}\n
                 xGOT Open Play: {player_row['xGOTop']:.2f}\n
                 Conversion Rate: {player_row['conversion rate (%)']:.2f}\n
                 Adj. SGA: {player_row['Adj. SGA']:.2f}\n
                 Shot Quality: {player_row['Shot Quality']:.3f}\n
                 """
             )

         player_row = df[df['Player Name'] == player_name].iloc[0]
         stat_text = player_stat_disp(player_row)

         # Bold the numbers in the stats
         bold_text = "\n".join([
             line.replace(f"{num}", r"$\bf{" + str(num) + "}$") if num else line
             for line in stat_text.split('\n')
             for num in [line.split(': ')[-1]] if num.replace('.', '', 1).isdigit()
         ])

         # Add the "Shot Quality Meter" text vertically next to the color bar
         fig.text(1.02, 0.5, 'Shot Quality Meter', va='center', rotation='vertical', fontsize=12, fontweight='bold')

         # Add the stats text to the plot (bottom left corner) with left alignment and reduced spacing
         ax.text(55, 5, bold_text, ha='left', va='bottom', fontsize=10, color='black', linespacing=1.25)

         plt.show()
```

This code generates a detailed visualization of a player's shot locations from open play on a half-pitch, using a convex hull to outline the shooting area. It begins by calculating the convex hull of the shot points to highlight the player's shooting range. The hull is then filled with a color representing the player's Shot Quality, using a colormap that varies based on the metric, with transparency controlled to emphasize differences. A color bar labeled "Shot Quality Meter" is added for reference. The plot also includes a title featuring the player's name and displays key performance statistics (e.g., goals, xG, xGOT) in a visually appealing format with bolded values for emphasis. This allows for quick interpretation of both the player's shooting efficiency and their shot distribution on the field. Overall, this visualization provides an in-depth analysis of a player's shooting patterns and effectiveness in open play.

The resulting visualization includes a Shot Quality Meter provides a color-coded reference for evaluating shot quality. The meter ranges from red to blue, with more blue indicating higher shot quality. A higher shot quality suggests that the player's shots are well-placed, often in areas of the goal that are more difficult for the goalkeeper to save. This metric helps in assessing not just the quantity of shots, but their effectiveness and precision, providing a deeper insight into the player's scoring potential.



90s Played: **20.80**
Total Goals: **15**
Open Play Goals: **13**
Total Shots: **78**
Shots on Target: **39**
xG: **13.41**
xG Open Play: **10.85**
xGOT: **15.74**
xGOT Open Play: **13.07**
Conversion Rate: **38.46**
Adj. SGA: **2.22**
Shot Quality: **0.363**

The next step is to fully automate the web scraping process, transforming this project into a fully functional backend capable of extracting player shot map data from various seasons across their careers. This backend would support a user-friendly frontend, where users can input a player's name and season to instantly generate visualizations. These visualizations would be accompanied by detailed explanations, showcasing the insights and research developed over the years.