Analyzing And-1 Plays on Historical Team and Single Player Performance in the NBA from 1996-2023

Andrew Boulle
andyboulle@gmail.com
Brendan Sacks
brendansacks21@gmail.com

MATH498R Final Project
Spring 2023
Prof. Yanir A. Rubinstein

May 18, 2023

Contents

Abstract

The goal of our project was to analyze "and 1" plays in the NBA. This is a play where the player shooting gets fouled by the opposing team, and still makes the basket. This results in the shooter being awarded 1 free throw on top of the made basket. This play is often referred to as an "and 1". In order to analyze these plays, we examined play by play data from the NBA API dating back to the 1996-1997 season, as that was the first season the NBA API began keeping a log of play by play data from each game. (DISCLAIMER: when anything is mentioned as being the best/worst of "all time" it means since the 1996-1997 season, since that is when the play by play data began being tracked.) From this data, we aimed to uncover a few things about and-1 plays: who the best players historically are at drawing and-1's, which players had the most and-1's in a single season, which team's were the best and worst at both drawing and committing and-1's, and how these and-1 plays affected offensive performance, defensive performance, and overall team success.

1 Code

1.1 csv_maker.ipynb

1. `from nba_api.stats.endpoints import PlayerDashboardByYearOverYear`
Imports PlayerDashboardByYearOverYear into out program. Used to fetch player stats, broken down by year.

2. `from nba_api.stats.static import players`
Imports the players module from nba_api.stats.static. This module provides information about NBA players.

3. `from nba_api.stats.endpoints import PlayerCareerStats`
Imports the PlayerCareerStats class from the nba_api package. This class is used to fetch career stats for a specific player.

4. `from nba_api.stats.endpoints import PlayByPlayV2 as pbp2`
This line imports the PlayByPlayV2 class (and renames it as pbp2) from the nba_api package. This class is used to fetch detailed play-by-play information for a specific NBA game.

5. `from nba_api.stats.endpoints import leaguegamefinder`
This line imports the leaguegamefinder class from the nba_api package. This class is used to find specific games based on various parameters such as team, season, etc.

6. `from nba_api.stats.static import teams`
This line imports the teams module from nba_api.stats.static. This module provides static information about NBA teams.

7. `from nba_api.stats.endpoints import TeamGameLogs`
This line imports the TeamGameLogs class from the nba_api package. This class is used to fetch game logs for a specific NBA team.

8. `from nba_api.stats.endpoints import TeamYearByYearStats`
This line imports the TeamYearByYearStats class from the nba_api package. This class is used to fetch team statistics, broken down year by year.

9. `from datetime import datetime`
This line imports the datetime class from the datetime module. This class is often used to work with dates and times.

10. `import pandas as pd`
This line imports the pandas library (and renames it as pd). pandas is a popular data analysis and manipulation library in Python.

11. `import time`
This line imports the time module. This module provides various time-related functions.

12. `import re`

This line imports the re module. This module provides regular expression matching operations.

13. `import random`
This line imports the random module. This module implements pseudo-random number generators for various distributions.

14. `import requests`
This line imports the requests library. requests is a popular Python library for making HTTP requests.

15. `import os`
This line imports the os module. This module provides a way of using operating system dependent functionality, such as reading or writing to the file system.

16. `def retry(func, retries=3):`
Defines the retry function, with 3 retries before the function fails.

17. `def retry_wrapper(*args, **kwargs):`
This line defines a nested function retry_wrapper. This function will be used to wrap the original function func.

18. `attempts = 0`
This line initializes a variable attempts to 0. This variable keeps track of the number of attempts made to execute func.

19. `while attempts < retries:`
This line begins a while loop that will continue to iterate as long as the number of attempts is less than the maximum number of retries.

20. `try:`
This line begins a try block. The code within this block is executed, and if any exceptions are thrown, they are handled by the subsequent except block.

21. `return func(*args, **kwargs)`
This line calls the original function func. If func executes successfully, its return value is returned immediately from retry_wrapper.

22. `except requests.exceptions.RequestException as e:`
This line begins an except block that catches RequestException exceptions thrown by the requests library. Happens if the nba_api times out.

23. `print(e)`
Prints the exception that was thrown.

24. `time.sleep(0.700)`
Makes the program wait for 0.7 seconds to try and stop server overload.

25. `attempts += 1`
Add an attempt if the program does not work. If this reaches 3, then the program stops running.

26. `return retry_wrapper`
Returns the function, indicating it has come to an end.

27. `@retry`
This is a decorator that will repeat the function it is attached to in case of an exception, until it executes successfully.

28. `def getSingleGamePBP(game_id):`
This line defines a function named getSingleGamePBP that takes one parameter, game_id.

29. `full_single_game_pbp = pbp2(game_id=game_id)`
This line calls the pbp2 function with the game_id parameter and assigns the resulting play by play data to the variable full_single_game_pbp.

30. `full_single_game_pbp = full_single_game_pbp.get_data_frames()[0]`
This line calls the get_data_frames method on the full_single_game_pbp object, which returns the play by play data by itself.

31. `return full_single_game_pbp`
This line returns the full_single_game_pbp data frame from the function.

32. `@retry`
This is a decorator that will repeat the function it is attached to in case of an exception, until it executes successfully.

33. `def getFullSeasonPBP(season_string):`
This line defines a function named getFullSeasonPBP that takes one parameter, season_string, which is a string representation of the season year.

34. `print('Retrieving Game IDs for the ' + season_string + ' season at ' + str(datetime.now()))`
This line prints a message to the console indicating that the retrieval of Game IDs for the given season has started, along with the current time.

35. `single_season_log = TeamGameLogs(season_nullable=season_string, season_type_nullable='Regular Season', league_id_nullable='00')`
This line calls the TeamGameLogs function to get the game logs for a single season of regular season games in the NBA. The results are assigned to the variable single_season_log.

36. `single_season_logs = single_season_log.get_data_frames()[0]`
Gets the actual log data.

37. `print('Finished retrieving Game IDs for the ' + season_string + ' season at ' + str(datetime.now()))`
This line prints a message to the console indicating that the retrieval of Game IDs for the given season has completed, along with the current time.

38. `unique_games = single_season_logs.drop_duplicates(subset='GAME_ID')`
This line calls the drop_duplicates method on the single_season_logs DataFrame to remove duplicate games based on the GAME_ID column. The resulting DataFrame is assigned to unique_games.

39. `single_season_game_ids = unique_games['GAME_ID']`
This line selects the GAME_ID column from the unique_games DataFrame and assigns it to single_season_game_ids.

40. `single_season_pbp_df_list = []`
This line initializes an empty list single_season_pbp_df_list. This list will be used to store play-by-play data for each game.

41. `i = 0`
This line initializes a counter i to 0. This counter will be used to track the progress of the game data retrieval.

42. `total = len(single_season_game_ids)`
This line gets the number of game IDs in a single season and assigns it to total.

43. `for game_id in single_season_game_ids:`
This line starts a for loop that iterates over each game ID in the single_season_game_ids Series.

44. `i += 1`
This line increments the counter i by 1 in each iteration of the loop.

45. `print(f"Progress: 100 * i/total: .2f%", end='\r')`
This line prints the progress of the data retrieval process as a percentage to the console.

46. `full_single_game_pbp = getSingleGamePBP(game_id)`
This line calls the getSingleGamePBP function with the current game ID and assigns the result to full_single_game_pbp.

47. `single_season_pbp_df_list.append(full_single_game_pbp)`
This line appends the DataFrame full_single_game_pbp to the list single_season_pbp_df_list.

48. `single_season_pbp_df = pd.concat(single_season_pbp_df_list)`
This line concatenates all the DataFrames in the list single_season_pbp_df_list into a single DataFrame and assigns it to single_season_pbp_df.

49. `file_path = season_string + '.csv'`
This line constructs a file path for saving the resulting DataFrame to a CSV file.

50. `single_season_pbp_df.to_csv(file_path, index=False)`
This line writes the DataFrame single_season_pbp_df to a CSV file at the specified file path. The index parameter is set to False, so the DataFrame's index will not be included in the CSV file.

51. `return single_season_pbp_df`
This line returns the DataFrame single_season_pbp_df from the function. This DataFrame contains the play-by-play data for the entire season.

52. `all_time_pbp_df_list = []`
Initializes an empty list that will be used to store the play-by-play data for all seasons.

53. `season_string = ""`
Initializes an empty string that will be used to construct the string representation of the season years.

55. `for i in range(96, 99):`
Starts a loop iterating through the years 1996 to 1998, using the variable i to form the season string.

56. `season_string = "19" + str(i) + "-" + str(i + 1)`
Constructs the season string for the current iteration, concatenating the years (e.g., "1996-97").

57. `print('Retrieving Play by Play data for the ' + season_string + ' season at ' + str(datetime.now()))`
Prints a message to the console to inform the user that the play-by-play data for the current season is being retrieved, along with the current timestamp.

58. `single_season_pbp_df = getFullSeasonPBP(season_string)`
Calls the getFullSeasonPBP function with the current season_string and stores the resulting play-by-play DataFrame in the variable single_season_pbp_df.

59. `all_time_pbp_df_list.append(single_season_pbp_df)`
Appends the DataFrame containing the play-by-play data for the current season to the all_time_pbp_df_list.

60. `print('Finished retrieving Play by Play data for the ' + season_string + ' season at ' + str(datetime.now()))`
Prints a message to the console to inform the user that the play-by-play data retrieval for the current season has been completed, along with the current timestamp.

61. `season_string = "1999-00"`
Sets the season_string to 1999-00 to denote the specific case of the 1999-2000 season.

62. `print('Retrieving Play by Play data for the ' + season_string + ' season at ' + str(datetime.now()))`
Prints a status message to the console, indicating that the play-by-play data for the 1999-00 season is being retrieved, along with the current timestamp.

63. `single_season_pbp_df = getFullSeasonPBP(season_string)`
Calls the getFullSeasonPBP function with the season_string 1999-00, storing the resulting DataFrame in single_season_pbp_df.

64. `all_time_pbp_df_list.append(single_season_pbp_df)`
Appends the single_season_pbp_df DataFrame to the all_time_pbp_df_list.

65. `print('Finished retrieving Play by Play data for the ' + season_string + ' season at ' + str(datetime.now()))`
Prints a status message to the console, indicating that the retrieval of play-by-play data for the 1999-00 season has been completed, along with the current timestamp.

67. `for i in range(0, 9):`
Starts a loop that iterates from 0 to 8, where i is used to form the season string for the years 2000 to 2009.

68. `season_string = "200" + str(i) + "-0" + str(i + 1)`
Constructs the season string for the current iteration, concatenating the years (e.g., "2000-01").

`69. print('Retrieving Play by Play data for the ' + season_string + ' season at ' + str(datetime.now()))`
Prints a status message to the console, indicating that the play-by-play data for the current season (from 2000 to 2009) is being retrieved, along with the current timestamp.

`70. single_season_pbp_df = getFullSeasonPBP(season_string)`
Calls the getFullSeasonPBP function with the current season_string and stores the resulting play-by-play DataFrame in the variable single_season_pbp_df.

`71. all_time_pbp_df_list.append(single_season_pbp_df)`
Appends the DataFrame containing the play-by-play data for the current season to the all_time_pbp_df_list.

`72. print('Finished retrieving Play by Play data for the ' + season_string + ' season at ' + str(datetime.now()))`
Prints a message to the console to inform the user that the play-by-play data retrieval for the current season has been completed, along with the current timestamp.
`73. season_string = "2009-10"`
Sets the season_string to 2009-10 to denote the specific case of the 2009-2010 season.

`74. print('Retrieving Play by Play data for the ' + season_string + ' season at ' + str(datetime.now()))`
Prints a status message to the console, indicating that the play-by-play data for the 2009-10 season is being retrieved, along with the current timestamp.

`75. single_season_pbp_df = getFullSeasonPBP(season_string)`
Calls the getFullSeasonPBP function with the season_string 2009-10, storing the resulting DataFrame in single_season_pbp_df.

`76. all_time_pbp_df_list.append(single_season_pbp_df)`
Appends the single_season_pbp_df DataFrame to the all_time_pbp_df_list.

`77. print('Finished retrieving Play by Play data for the ' + season_string + ' season at ' + str(datetime.now()))`
Prints a status message to the console, indicating that the retrieval of play-by-play data for the 2009-10 season has been completed, along with the current timestamp.

`79. for i in range(0, 9):`
Starts a loop that iterates from 0 to 8, where i is used to form the season string for the years 2010 to 2019.

`80. season_string = "201" + str(i) + "-1" + str(i + 1)`
Constructs the season string for the current iteration, concatenating the years (e.g., "2010-11").

`81. print('Retrieving Play by Play data for the ' + season_string + ' season at ' + str(datetime.now()))`
Prints a status message to the console, indicating that the play-by-play data for the current season (from 2010 to 2019) is being retrieved, along with the current timestamp.

`82. single_season_pbp_df = getFullSeasonPBP(season_string)`
Calls the getFullSeasonPBP function with the current season_string and stores the resulting play-by-play DataFrame in the variable single_season_pbp_df.

`83. all_time_pbp_df_list.append(single_season_pbp_df)`
Appends the DataFrame containing the play-by-play data for the current season to the all_time_pbp_df_list.

`84. print('Finished retrieving Play by Play data for the ' + season_string + ' season at ' + str(datetime.now()))`
Prints a message to the console to inform the user that the play-by-play data retrieval for the current season has been completed, along with the current timestamp.
`85. season_string = "2019-20"`
Sets the season_string to "2019-20" to denote the specific case of the 2019-2020 season.

`86. print('Retrieving Play by Play data for the ' + season_string + ' season at ' + str(datetime.now()))`

Prints a status message to the console, indicating that the play-by-play data for the 2019-20 season is being retrieved, along with the current timestamp.

```
87.  single_season_pbp_df = getFullSeasonPBP(season_string)
```
Calls the getFullSeasonPBP function with the season_string 2019-20, storing the resulting DataFrame in single_season_pbp_df.

```
88.  all_time_pbp_df_list.append(single_season_pbp_df)
```
Appends the single_season_pbp_df DataFrame to the all_time_pbp_df_list.

```
89.  print('Finished retrieving Play by Play data for the ' + season_string + ' season
at ' + str(datetime.now()))
```
Prints a status message to the console, indicating that the retrieval of play-by-play data for the 2019-20 season has been completed, along with the current timestamp.

```
91.  for i in range(0, 3):
```
Starts a loop that iterates from 0 to 2, where i is used to form the season string for the years 2020 to 2023.

```
92.  season_string = "202" + str(i) + "-2" + str(i + 1)
```
Constructs the season string for the current iteration, concatenating the years (e.g., "2020-21").

```
93.  print('Retrieving Play by Play data for the ' + season_string + ' season at ' +
str(datetime.now()))
```
Prints a status message to the console, indicating that the play-by-play data for the current season (from 2020 to 2023) is being retrieved, along with the current timestamp.

```
94.  single_season_pbp_df = getFullSeasonPBP(season_string)
```
Calls the getFullSeasonPBP function with the current season_string and stores the resulting play-by-play DataFrame in the variable single_season_pbp_df.

```
95.  all_time_pbp_df_list.append(single_season_pbp_df)
```
Appends the DataFrame containing the play-by-play data for the current season to the all_time_pbp_df_list.

```
96.  print('Finished retrieving Play by Play data for the ' + season_string + ' season
at ' + str(datetime.now()))
```
Prints a message to the console to inform the user that the play-by-play data retrieval for the current season has been completed, along with the current timestamp.

```
97.  all_time_pbp_df = pd.concat(all_time_pbp_df_list)
```
Combines all the DataFrames in the all_time_pbp_df_list into a single DataFrame all_time_pbp_df using the pd.concat function.

```
98.  file_path = 'allpbp.csv'
```
Sets the file_path string to the specific location where the final CSV file will be saved.

```
99.  all_time_pbp_df.to_csv(file_path, index=False)
```
Writes the all_time_pbp_df DataFrame to a CSV file at the specified file_path. The index=False argument means that the DataFrame index will not be written to the file.

```
100.  allpbp_file_path = 'allpbp.csv'
```
Sets the allpbp_file_path string to the specific location where the final CSV file will be read from.

```
101.  pbp_df = pd.read_csv(allpbp_file_path)
```
Reads the CSV file at the specified allpbp_file_path into a DataFrame pbp_df using the pd.read_csv function.

```
102.  columns_to_keep = ['GAME_ID', 'EVENTMSGTYPE', 'EVENTMSGACTIONTYPE', 'HOMEDESCRIPTION',
'NEUTRALDESCRIPTION', 'VISITORDESCRIPTION', 'PLAYER1_ID', 'PLAYER1_NAME', 'PLAYER1_TEAM_ID',
'PLAYER1_TEAM_CITY', 'PLAYER1_TEAM_NICKNAME', 'PLAYER1_TEAM_ABBREVIATION', 'PLAYER2_ID',
'PLAYER2_NAME', 'PLAYER2_TEAM_ID', 'PLAYER2_TEAM_CITY', 'PLAYER2_TEAM_NICKNAME',
'PLAYER2_TEAM_ABBREVIATION']
```
Defines a list of column names columns_to_keep that will be kept in the DataFrame.

```
103.  pbp_df_condensed = pbp_df.filter(columns_to_keep)
```

Creates a new DataFrame pbp_df_condensed that contains only the columns specified in the columns_to_keep list from the pbp_df DataFrame.

104. `file_path_filtered = 'allpbp_filtered.csv'`
Sets the file_path_filtered string to the specific location where the filtered CSV file will be saved.

105. `pbp_df_condensed.to_csv(file_path_filtered, index=False)`
Writes the pbp_df_condensed DataFrame to a CSV file at the specified file_path_filtered. The index=False argument means that the DataFrame index will not be written to the file.

106. `allpbp_file_path = 'allpbp_filtered.csv'`
Sets the allpbp_file_path string to the specific location where the filtered CSV file will be read from.

107. `pbp_df = pd.read_csv(allpbp_file_path)`
Reads the CSV file at the specified allpbp_file_path into a DataFrame pbp_df using the pd.read_csv function.

108. `all_time_shots_and_fouls_pbp = pbp_df[(pbp_df['EVENTMSGTYPE'] == 1) | (pbp_df['EVENTMSGTYPE'] == 6) | ((pbp_df['EVENTMSGTYPE'] == 3) & ((pbp_df['EVENTMSGACTIONTYPE'] == 10) | (pbp_df['EVENTMSGACTIONTYPE'] == 11) | (pbp_df['EVENTMSGACTIONTYPE'] == 13)))]`
Creates a new DataFrame all_time_shots_and_fouls_pbp that contains only the rows from pbp_df where the EVENTMSGTYPE is either 1, 6, or (3 and EVENTMSGACTIONTYPE is 10, 11, or 13). Shot, shooting foul, or foul shot.

109. `columns_to_keep = ['GAME_ID', 'EVENTMSGTYPE', 'EVENTMSGACTIONTYPE', 'HOMEDESCRIPTION', 'VISITORDESCRIPTION', 'PLAYER1_ID', 'PLAYER1_NAME', 'PLAYER1_TEAM_ABBREVIATION', 'PLAYER2_ID', 'PLAYER2_NAME', 'PLAYER2_TEAM_ABBREVIATION']`
Defines a list of column names columns_to_keep that will be kept in the DataFrame.

110. `all_time_shots_and_fouls_pbp = all_time_shots_and_fouls_pbp.filter(columns_to_keep)`
Creates a new DataFrame all_time_shots_and_fouls_pbp that contains only the columns specified in the columns_to_keep list from the all_time_shots_and_fouls_pbp DataFrame.

111. `all_time_shots_and_fouls_pbp.head()`
Displays the first 5 rows of the all_time_shots_and_fouls_pbp DataFrame.

112. `df = all_time_shots_and_fouls_pbp`
Assigns the DataFrame all_time_shots_and_fouls_pbp to a new variable df for ease of use in further processing.

113. `and1_df = pd.DataFrame(columns=['Game_ID', 'Player_ID', 'Team', 'Opposing_Team', 'And_1s', 'Shooting_Fouls','And_1s_2pt', 'And_1s_3pt', 'Shooting_Fouls_2pt', 'Shooting_Fouls_3pt'])`
Creates a new empty DataFrame and1_df with specified column names. This DataFrame will hold data related to And-1s.

114. `total = len(df)`
Stores the total number of rows in df into the variable total.

115. `for i in range(2, total):`
Starts a loop from 2 to the total number of rows in the DataFrame df. The loop iterates over each row of the DataFrame for processing.

116. `print(f"Progress: 100 * i/(total): .2f%", end='\r')`
Prints the progress of the loop as a percentage of the total number of rows in the DataFrame. This is useful for long-running tasks.

117. `if df['EVENTMSGTYPE'].iloc[i - 1] == 6 and df['EVENTMSGACTIONTYPE'].iloc[i - 1] == 2:`
Checks if the EVENTMSGTYPE of the previous row (i-1) is 6 and the EVENTMSGACTIONTYPE is 2, indicating a shooting foul.

118. `if df['EVENTMSGTYPE'].iloc[i] == 3 and df['EVENTMSGACTIONTYPE'].iloc[i] == 10:`
If the previous condition is met, checks if the EVENTMSGTYPE of the current row (i) is 3 and the EVENTMSGACTIONTYPE is 10, indicating a 1 shot Free Throw line visit.

```
119.  if df['EVENTMSGTYPE'].iloc[i - 2] == 1 and df['PLAYER1_ID'].iloc[i] == df['PLAYER1_ID'].iloc[i
- 2]:
```
If the previous condition is met, checks if the EVENTMSGTYPE two rows prior (i-2) is 1 and the PLAYER1_ID of the current row (i) matches the PLAYER1_ID two rows prior, indicating a made basket by the same player who was fouled.

```
120.  if re.search('3PT', str(df['HOMEDESCRIPTION'].iloc[i - 2])) or re.search('3PT',
str(df['VISITORDESCRIPTION'].iloc[i - 2])):
```
If the previous condition is met, checks if the HOMEDESCRIPTION or VISITOR two rows prior contains the string '3PT', indicating a 3-point shot.

```
121.  and1_df.loc[len(and1_df)] = [df['GAME_ID'].iloc[i], df['PLAYER1_ID'].iloc[i],
df['PLAYER1_TEAM_ABBREVIATION'].iloc[i], df['PLAYER1_TEAM_ABBREVIATION'].iloc[i-1], 1,
1, 0, 1, 0, 1]
```
If the previous condition is met, adds a new row to the DataFrame and1_df with the corresponding values, indicating a successful 3-point And-1 play.

```
122.  else:
```
If the previous condition is not met (i.e., the shot was not a 3-point shot), executes the following block of code.

```
123.  and1_df.loc[len(and1_df)] = [df['GAME_ID'].iloc[i], df['PLAYER1_ID'].iloc[i],
df['PLAYER1_TEAM_ABBREVIATION'].iloc[i], df['PLAYER1_TEAM_ABBREVIATION'].iloc[i-1], 1,
1, 1, 0, 1, 0]
```
Adds a new row to the DataFrame and1_df with the corresponding values, indicating a successful 2-point And-1 play.

```
124.  else:
```
If the shooting foul did not result in an And-1, executes the following block of code.

```
125.  if df['EVENTMSGTYPE'].iloc[i] == 3 and df['EVENTMSGACTIONTYPE'].iloc[i] == 13:
```
Checks if the EVENTMSGTYPE of the current row (i) is 3 and the EVENTMSGACTIONTYPE is 13, indicating a shooting foul on a 3-point attempt.

```
126.  and1_df.loc[len(and1_df)] = [df['GAME_ID'].iloc[i], df['PLAYER1_ID'].iloc[i],
df['PLAYER1_TEAM_ABBREVIATION'].iloc[i], df['PLAYER1_TEAM_ABBREVIATION'].iloc[i-1], 0,
1, 0, 0, 0, 1]
```
If the previous condition is met, adds a new row to the DataFrame and1_df with the corresponding values, indicating a shooting foul on a 3-point attempt without an And-1.

```
127.  elif df['EVENTMSGTYPE'].iloc[i] == 3 and df['EVENTMSGACTIONTYPE'].iloc[i] == 11:
```
Checks if the EVENTMSGTYPE of the current row (i) is 3 and the EVENTMSGACTIONTYPE is 11, indicating a shooting foul on a 2-point attempt.

```
128.  and1_df.loc[len(and1_df)] = [df['GAME_ID'].iloc[i], df['PLAYER1_ID'].iloc[i],
df['PLAYER1_TEAM_ABBREVIATION'].iloc[i], df['PLAYER1_TEAM_ABBREVIATION'].iloc [i-1], 0,
1, 0, 0, 1, 0]
```
If the previous condition is met, adds a new row to the DataFrame and1_df with the corresponding values, indicating a shooting foul on a 2-point attempt without an And-1.

```
129.  file_path = 'and1_df.csv'
```
Specifies the file path where the resulting DataFrame will be saved.

```
130.  and1_df.to_csv(file_path, index=False)
```
Writes the and1_df DataFrame to a CSV file at the specified file path, without including the DataFrame's index.

```
131.  file_path = 'and1_df.csv'
```
Specifies the file path where the and1_df.csv file is located.

```
132.  and1_df = pd.read_csv(file_path)
```
Reads the CSV file at the specified file path into a DataFrame called and1_df.

```
133.  season_and_game_ids_dict = dict()
```

Creates an empty dictionary called season_and_game_ids_dict to store the season-year and corresponding list of game IDs.

`for i in range(96, 99):`
Begins a loop from 1996 to 1999, iterating over each year.

`season_string = "19" + str(i) + "-" + str(i + 1)`
Generates a string representation of the current season by concatenating the year with its next year, e.g., "1996-1997".

`print('Retrieving Game IDs for the ' + season_string + ' season')`
Prints a message indicating the season for which game IDs are being retrieved.

`result = TeamGameLogs(season_nullable=season_string, league_id_nullable='00')`
Retrieves game logs for the current season using the TeamGameLogs function.

`game_ids = result.get_data_frames()[0]['GAME_ID'].tolist()`
Extracts the game IDs from the first data frame of the result object and converts them to a list.

`season_and_game_ids_dict[season_string] = game_ids`
Adds an entry to the season_and_game_ids_dict dictionary, using the current season as the key and the list of game IDs as the value.

`season_string = "1999-00"`
Sets the season_string variable to "1999-00" for the next block of code.

`print('Retrieving Game IDs for the ' + season_string + ' season')`
Prints a message indicating the season for which game IDs are being retrieved.

`result = TeamGameLogs(season_nullable=season_string, league_id_nullable='00')`
Retrieves game logs for the season "1999-00" using the TeamGameLogs function.

`game_ids = result.get_data_frames()[0]['GAME_ID'].tolist()`
Extracts the game IDs from the first data frame of the result object and converts them to a list.

`season_and_game_ids_dict[season_string] = game_ids`
Adds an entry to the season_and_game_ids_dict dictionary for the season "1999-00" with the list of game IDs.

`for i in range(0, 9):`
Begins a loop from 0 to 9, iterating over each value. Iterating through 2000-2009 seasons.

`season_string = "200" + str(i) + "-0" + str(i + 1)`
Generates a string representation of the current season by concatenating the year with its next year, e.g., "2000-2001".

`print('Retrieving Game IDs for the ' + season_string + ' season')`
Prints a message indicating the season for which game IDs are being retrieved.

`result = TeamGameLogs(season_nullable= season_string, league_id_nullable='00')`
Retrieves game logs for the current season using the TeamGameLogs function.

`game_ids = result.get_data_frames()[0]['GAME_ID'].tolist()`
Extracts the game IDs from the first data frame of the result object and converts them to a list.

`season_and_game_ids_dict[season_string] = game_ids`
Adds an entry to the season_and_game_ids_dict dictionary, using the current season as the key and the list of game IDs as the value.

`season_string = "2009-10"`
Sets the season_string variable to "2009-10" for the next block of code.

`print('Retrieving Game IDs for the ' + season_string + ' season')`
Prints a message indicating the season for which game IDs are being retrieved.

`result = TeamGameLogs(season_nullable=season_string, league_id_nullable='00')`
Retrieves game logs for the season "2009-10" using the TeamGameLogs function.

154. `game_ids = result.get_data_frames()[0]['GAME_ID'].tolist()`
Extracts the game IDs from the first data frame of the result object and converts them to a list.

155. `season_and_game_ids_dict[season_string] = game_ids`
Adds an entry to the season_and_game_ids_dict dictionary for the season "2009-10" with the list of game IDs.

156. `for i in range(0, 9):`
Begins a loop from 0 to 9, iterating over each value. Iterating through 2010-2019 seasons.

157. `season_string = "201" + str(i) + "-1" + str(i + 1)`
Generates a string representation of the current season by concatenating the year with its next year, e.g., "2010-2011".

158. `print('Retrieving Game IDs for the ' + season_string + ' season')`
Prints a message indicating the season for which game IDs are being retrieved.

159. `result = TeamGameLogs(season_nullable= season_string, league_id_nullable='00')`
Retrieves game logs for the current season using the TeamGameLogs function.

160. `game_ids = result.get_data_frames()[0]['GAME_ID'].tolist()`
Extracts the game IDs from the first data frame of the result object and converts them to a list.

161. `season_and_game_ids_dict[season_string] = game_ids`
Adds an entry to the season_and_game_ids_dict dictionary, using the current season as the key and the list of game IDs as the value.

162. `season_string = "2019-20"`
Sets the season_string variable to "2019-20" for the next block of code.

163. `print('Retrieving Game IDs for the ' + season_string + ' season')`
Prints a message indicating the season for which game IDs are being retrieved.

164. `result = TeamGameLogs(season_nullable=season_string, league_id_nullable='00')`
Retrieves game logs for the season "2019-20" using the TeamGameLogs function.

165. `game_ids = result.get_data_frames()[0]['GAME_ID'].tolist()`
Extracts the game IDs from the first data frame of the result object and converts them to a list.

166. `season_and_game_ids_dict[season_string] = game_ids`
Adds an entry to the season_and_game_ids_dict dictionary for the season "2009-10" with the list of game IDs.

167. `for i in range(0, 3):`
Begins a loop from 0 to 3, iterating over each value. Iterating through 2020-2023 seasons.

168. `season_string = "202" + str(i) + "-2" + str(i + 1)`
Generates a string representation of the current season by concatenating the year with its next year, e.g., "2020-2021".

169. `print('Retrieving Game IDs for the ' + season_string + ' season')`
Prints a message indicating the season for which game IDs are being retrieved.

170. `result = TeamGameLogs(season_nullable= season_string, league_id_nullable='00')`
Retrieves game logs for the current season using the TeamGameLogs function.

171. `game_ids = result.get_data_frames()[0]['GAME_ID'].tolist()`
Extracts the game IDs from the first data frame of the result object and converts them to a list.

172. `season_and_game_ids_dict[season_string] = game_ids`
Adds an entry to the season_and_game_ids_dict dictionary, using the current season as the key and the list of game IDs as the value.

173. `and1_df['Season'] = None`
This line adds a new column named Season to the DataFrame and1_df and initializes all its values to None.

174. `total = len(and1_df)`
This line assigns the number of rows in the DataFrame and1_df to the variable "total".

175. `for i in range(2, total):`
This line starts a loop that will iterate over the range from 2 to the total number of rows in the DataFrame.

176. `print(f"Progress: 100 * i/(total): .2f%", end='\r')`
This line prints the progress of the loop as a percentage to the console.

177. `row = and1_df.iloc[i]`
This line gets the i-th row of the DataFrame and1_df and assigns it to the variable row.

178. `game_id = row.Game_ID`
This line gets the value of the Game_ID field from the current row and assigns it to the variable game_id.

179. `s = '00' + str(game_id)`
This line converts the game ID to a string, prepends '00' to it, and assigns the result to the variable "s".

180. `for key in season_and_game_ids_dict:`
This line starts a nested loop that will iterate over the keys in the dictionary season_and_game_ids_dict.

181. `if s in season_and_game_ids_dict[key]:`
This line checks if the game ID string s is in the list of game IDs associated with the current key in the dictionary.

182. `and1_df.at[i, 'Season'] = key`
If the game ID is found in the list, this line assigns the key (which represents the season) to the Season field of the current row in the DataFrame.

183. `file_path = 'and1_df_w_season.csv'`
This line constructs a file path for saving the updated DataFrame to a CSV file.

184. `and1_df.to_csv(file_path, index=False)`
This line writes the DataFrame and1_df to a CSV file at the specified file path. The indexes will not be saved to the CSV.

185. `file_path = 'and1_df_w_season.csv'`
This line assigns a string representing a file path to the variable file_path. This file path is where the CSV file and1_df_w_season.csv is located.

186. `and1_df = pd.read_csv(file_path)`
This line reads the CSV file at the specified file path into a DataFrame and1_df.

187. `all_teams = teams.get_teams()`
This line calls the get_teams function which returns a list of all NBA teams and assigns this list to the variable all_teams.

188. `team_ids = [team['id'] for team in all_teams]`
This line iterates through each team in the all_teams list and collects each team's ID into a new list team_ids.

189. `all_teams_yearly_stats = pd.DataFrame()`
This line creates an empty DataFrame named all_teams_yearly_stats.

190. `starting_season = '1995-96'`
This line assigns the string '1995-96' to the variable starting_season, which will be used to filter out statistics from seasons earlier than this one.

191. `i = 0`
This line initializes a variable i to 0, which will be used to track the progress of the subsequent loop.

192. `total = len(team_ids)`
This line assigns the number of team IDs in the team_ids list to the variable total.

```
193.  for id in team_ids:
```
This line starts a loop that will iterate over each ID in the team_ids list.

```
194.  i += 1
```
This line increments the variable i by 1 with each iteration of the loop.

```
195.  print(f"Progress:  100 * i/total:  .2f%", end='\r')
```
This line prints the progress of the loop as a percentage to the console.

```
196.  team_year_stats = TeamYearByYearStats(team_id=id, league_id='00', season_type_all_star='Regular
Season', per_mode_simple='PerGame')
```
This line calls the TeamYearByYearStats function. The result is a DataFrame of the team's yearly statistics, which is assigned to the variable team_year_stats.

```
197.  team_year_stats = team_year_stats.get_data_frames()[0]
```
This line extracts the first DataFrame from team_year_stats, and reassigns it to team_year_stats.

```
198.  filtered_stats = team_year_stats[team_year_stats['YEAR'] > starting_season]
```
This line filters team_year_stats to include only the rows where the 'YEAR' is greater than starting_season, and assigns the resulting DataFrame to filtered_stats.

```
199.  all_teams_yearly_stats = pd.concat([all_teams_yearly_stats, filtered_stats])
```
This line concatenates the filtered_stats DataFrame to the end of all_teams_yearly_stats, updating it with the new data.

```
200.  file_path = 'team_yearly_stats.csv'
```
This line assigns a new file path to the variable file_path.

```
201.  all_teams_yearly_stats.to_csv(file_path, index=False)
```
This line saves all_teams_yearly_stats as a CSV file at the specified file_path, without including the DataFrame's index.

1.2 498R_Final.ipynb

1. `from nba_api.stats.static import players`
Imports the players module from nba_api.stats.static. This module provides information about NBA players.

2. `from nba_api.stats.static import teams`
Imports the teams module from nba_api.stats.static. This module provides information about NBA teams.

3. `import pandas as pd`
Imports the pandas library and assigns it the alias "pd". This library is used for data manipulation and analysis.

4. `import matplotlib.pyplot as plt`
Imports the pyplot module from the matplotlib library and assigns it the alias "plt". This module is used for creating plots and visualizations.

5. `import scipy.stats as stats`
Imports the stats module from the scipy library and assigns it the alias "stats". This module provides various statistical functions and distributions.

6. `pd.set_option('display.max_columns', 50)`
Sets the maximum number of columns to display in pandas dataframes to 50.

7. `pd.set_option('display.max_rows', 999)`
Sets the maximum number of rows to display in pandas dataframes to 999.

8. `and1_df = pd.read_csv('and1_df_w_season.csv')`
Reads the CSV file located at 'and1_df_w_season.csv' and assigns the resulting DataFrame to the variable 'and1_df'.

9. `player_and1_attempts = and1_df.groupby(['Player_ID']).sum(numeric_only=True)`
Groups all and-1 plays by player only using the 'Player_ID' column. The 'sum' function is applied to numeric columns only. The resulting grouped data is assigned to the variable 'player_and1_attempts'.

10. `player_and1_attempts.reset_index(inplace=True)`
Resets the index of the 'player_and1_attempts' DataFrame. The 'inplace=True' argument modifies the DataFrame directly, without creating a new object.

11. `all_time_and1_leaders_total = player_and1_attempts.sort_values(by=['And_1s', 'Shooting_Fouls'], ascending=[False, True]).head(20)`
Calculates the total and-1 leaders of all time by sorting the 'player_and1_attempts' DataFrame based on the 'And_1s' and 'Shooting_Fouls' columns in descending and ascending order, respectively. The top 20 rows are selected and assigned to the variable 'all_time_and1_leaders_total'.

12. `all_time_and1_leaders_total['Player Name'] = all_time_and1_leaders_total['Player_ID'].apply(players`
Adds a new column 'Player Name' to the 'all_time_and1_leaders_total' DataFrame by applying the 'players.find_player_by_id' function to the 'Player_ID' column and extracting the 'full_name' property.

13. `all_time_and1_leaders_total[['Player Name', 'And_1s', 'Shooting_Fouls', 'And_1s_2pt', 'And_1s_3pt']]`
Selects specific columns ('Player Name', 'And_1s', 'Shooting_Fouls', 'And_1s_2pt', 'And_1s_3pt') from the 'all_time_and1_leaders_total' DataFrame.

14. `all_time_and1_leaders_2pt = player_and1_attempts.sort_values(by=['And_1s_2pt', 'Shooting_Fouls'], ascending=[False, True]).head(20)`
Calculates the 2pt and-1 leaders of all time by sorting the 'player_and1_attempts' DataFrame based on the 'And_1s_2pt' and 'Shooting_Fouls' columns in descending and ascending order, respectively. The top 20 rows are selected and assigned to the variable 'all_time_and1_leaders_2pt'.

15. `all_time_and1_leaders_2pt['Player Name'] = all_time_and1_leaders_2pt['Player_ID'].apply(players.fi`
Adds a new column 'Player Name' to the 'all_time_and1_leaders_2pt' DataFrame by applying the 'players.find_player_by_id' function to the 'Player_ID' column and extracting the 'full_name' property.

16. `all_time_and1_leaders_2pt[['Player Name', 'And_1s_2pt', 'Shooting_Fouls', 'And_1s', 'And_1s_3pt']]`
Selects specific columns ('Player Name', 'And_1s_2pt', 'Shooting_Fouls', 'And_1s', 'And_1s_3pt') from the 'all_time_and1_leaders_2pt' DataFrame.

17. `all_time_and1_leaders_3pt = player_and1_attempts.sort_values(by=['And_1s_3pt', 'Shooting_Fouls'], ascending=[False, True]).head(20)`
Calculates the 3pt and-1 leaders of all time by sorting the 'player_and1_attempts' DataFrame based on the 'And_1s_3pt' and 'Shooting_Fouls' columns in descending and ascending order, respectively. The top 20 rows are selected and assigned to the variable 'all_time_and1_leaders_3pt'.

18. `all_time_and1_leaders_3pt['Player Name'] = all_time_and1_leaders_3pt['Player_ID'].apply(players.fi`
Adds a new column 'Player Name' to the 'all_time_and1_leaders_3pt' DataFrame by applying the 'players.find_player_by_id' function to the 'Player_ID' column and extracting the 'full_name' property.

19. `all_time_and1_leaders_3pt[['Player Name', 'And_1s_3pt', 'Shooting_Fouls', 'And_1s', 'And_1s_2pt']]`
Selects specific columns ('Player Name', 'And_1s_3pt', 'Shooting_Fouls', 'And_1s', 'And_1s_2pt') from the 'all_time_and1_leaders_3pt' DataFrame.

20. `player_and1_attempts_by_season = and1_df.groupby(['Player_ID', 'Season']).sum(numeric_only=True)`
Groups all and-1 plays by player and season using the 'Player_ID' and 'Season' columns. The 'sum' function is applied to numeric columns only. The resulting grouped data is assigned to the variable 'player_and1_attempts_by_season'.

21. `player_and1_attempts_by_season.reset_index(inplace=True)`
Resets the index of the 'player_and1_attempts_by_season' DataFrame. The 'inplace=True' argument modifies the DataFrame directly, without creating a new object.

22. `season_and1_leaders_total = player_and1_attempts_by_season.sort_values(by=['And_1s', 'Shooting_Fouls'], ascending=[False, True]).head(20)`
Calculates the total and-1 leaders by season by sorting the 'player_and1_attempts_by_season' DataFrame based on the 'And_1s' and 'Shooting_Fouls' columns in descending and ascending order, respectively. The top 20 rows are selected and assigned to the variable 'season_and1_leaders_total'.

23. `season_and1_leaders_total['Player Name'] = season_and1_leaders_total['Player_ID'].apply(players.fi`
Adds a new column 'Player Name' to the 'season_and1_leaders_total' DataFrame by applying the 'players.find_player_by_id' function to the 'Player_ID' column and extracting the 'full_name' property.

24. `season_and1_leaders_total[['Player Name', 'Season', 'And_1s', 'Shooting_Fouls', 'And_1s_2pt', 'And_1s_3pt']]`
Selects specific columns ('Player Name', 'Season', 'And_1s', 'Shooting_Fouls', 'And_1s_2pt', 'And_1s_3pt') from the 'season_and1_leaders_total' DataFrame.

25. `season_and1_leaders_2pt = player_and1_attempts_by_season.sort_values(by=['And_1s_2pt', 'Shooting_Fouls'], ascending=[False, True]).head(20)`
Calculates the 2pt and-1 leaders by season by sorting the 'player_and1_attempts_by_season' DataFrame based on the 'And_1s_2pt' and 'Shooting_Fouls' columns in descending and ascending order, respectively. The top 20 rows are selected and assigned to the variable 'season_and1_leaders_2pt'.

26. `season_and1_leaders_2pt['Player Name'] = season_and1_leaders_2pt['Player_ID'].apply(players.find_p`
Adds a new column 'Player Name' to the 'season_and1_leaders_2pt' DataFrame by applying the 'players.find_player_by_id' function to the 'Player_ID' column and extracting the 'full_name' property.

27. `season_and1_leaders_2pt[['Player Name', 'Season', 'And_1s_2pt', 'Shooting_Fouls', 'And_1s', 'And_1s_3pt']]`
Selects specific columns ('Player Name', 'Season', 'And_1s_2pt', 'Shooting_Fouls', 'And_1s', 'And_1s_3pt') from the 'season_and1_leaders_2pt' DataFrame.

28. `season_and1_leaders_3pt = player_and1_attempts_by_season.sort_values(by=['And_1s_3pt', 'Shooting_Fouls'], ascending=[False, True]).head(20)`

Calculates the 3pt and-1 leaders by season by sorting the 'player_and1_attempts_by_season' DataFrame based on the 'And_1s_3pt' and 'Shooting_Fouls' columns in descending and ascending order, respectively. The top 20 rows are selected and assigned to the variable 'season_and1_leaders_3pt'.

```
29. season_and1_leaders_3pt['Player Name'] = season_and1_leaders_3pt['Player_ID'].apply(players.find_p
```
Adds a new column 'Player Name' to the 'season_and1_leaders_3pt' DataFrame by applying the 'players.find_player_by_id' function to the 'Player_ID' column and extracting the 'full_name' property.

```
30. season_and1_leaders_3pt[['Player Name', 'Season', 'And_1s_3pt', 'Shooting_Fouls', 'And_1s',
'And_1s_2pt']]
```
Selects specific columns ('Player Name', 'Season', 'And_1s_3pt', 'Shooting_Fouls', 'And_1s', 'And_1s_2pt') from the 'season_and1_leaders_3pt' DataFrame.

```
31. all_time_total_and1_pct = and1_df['And_1s'].sum() / and1_df['Shooting_Fouls'].sum()
```
Calculates the total and-1 percentage by summing the 'And_1s' column in the 'and1_df' DataFrame and dividing it by the sum of the 'Shooting_Fouls' column. The result is assigned to the variable 'all_time_total_and1_pct'.

```
32. all_time_2pt_and1_pct = and1_df['And_1s_2pt'].sum() / and1_df['Shooting_Fouls_2pt'].sum()
```
Calculates the 2pt and-1 percentage by summing the 'And_1s_2pt' column in the 'and1_df' DataFrame and dividing it by the sum of the 'Shooting_Fouls_2pt' column. The result is assigned to the variable 'all_time_2pt_and1_pct'.

```
33. all_time_3pt_and1_pct = and1_df['And_1s_3pt'].sum() / and1_df['Shooting_Fouls_3pt'].sum()
```
Calculates the 3pt and-1 percentage by summing the 'And_1s_3pt' column in the 'and1_df' DataFrame and dividing it by the sum of the 'Shooting_Fouls_3pt' column. The result is assigned to the variable 'all_time_3pt_and1_pct'.

```
34. print('Shooting Fouls That Result in an And-1 Play:        '  + str((100 * all_time_total_and1_pc
+ '% (' + str(and1_df['And_1s'].sum()) + ' of ' + str(and1_df['Shooting_Fouls'].sum())
+ ')')
```
Prints the total and-1 percentage by formatting a string with the calculated value. The number is rounded to two decimal places using the 'round' function. The total number of 'And_1s' and 'Shooting_Fouls' in the 'and1_df' DataFrame are also displayed.

```
35. print('2pt Shooting Fouls That Result in an And-1 Play:      '  + str((100 * all_time_2pt_and1_pc
+ '% (' + str(and1_df['And_1s_2pt'].sum()) + ' of ' + str(and1_df['Shooting_Fouls_2pt'].sum())
+ ')')
```
Prints the 2pt and-1 percentage by formatting a string with the calculated value. The number is rounded to two decimal places using the 'round' function. The total number of 'And_1s_2pt' and 'Shooting_Fouls_2pt' in the 'and1_df' DataFrame are also displayed.

```
36. print('3pt Shooting Fouls That Result in an And-1 Play:      '  + str((100 * all_time_3pt_and1_pc
+ '% (' + str(and1_df['And_1s_3pt'].sum()) + ' of ' + str(and1_df['Shooting_Fouls_3pt'].sum())
+ ')')
```
Prints the 3pt and-1 percentage by formatting a string with the calculated value. The number is rounded to two decimal places using the 'round' function. The total number of 'And_1s_3pt' and 'Shooting_Fouls_3pt' in the 'and1_df' DataFrame are also displayed.

```
37. mask = player_and1_attempts['And_1s'] >= 100
```
Creates a boolean mask by comparing the 'And_1s' column of the 'player_and1_attempts' DataFrame with 100. This mask will be used to filter the DataFrame and select only players with 100 or more and-1 plays in their career.

```
38. player_and1_attempts_minimized = player_and1_attempts.loc[mask]
```
Creates a new DataFrame called 'player_and1_attempts_minimized' by selecting the rows from the 'player_and1_attempts' DataFrame where the corresponding values in the 'mask' are True. This filters the DataFrame to include only players with 100 or more and-1 plays in their career.

```
39. player_and1_attempts_minimized['And_1_Percentage'] = player_and1_attempts['And_1s']
/ player_and1_attempts['Shooting_Fouls']
```

Calculates the and-1 percentage by dividing the 'And_1s' column of the 'player_and1_attempts' DataFrame by the 'Shooting_Fouls' column. The result is assigned to a new column called 'And_1_Percentage' in the 'player_and1_attempts_minimized' DataFrame.

40. `player_and1_attempts_minimized['And_1_Percentage_2pt'] = player_and1_attempts['And_1s_2pt'] / player_and1_attempts['Shooting_Fouls_2pt']`
Calculates the 2pt and-1 percentage by dividing the 'And_1s_2pt' column of the 'player_and1_attempts' DataFrame by the 'Shooting_Fouls_2pt' column. The result is assigned to a new column called 'And_1_Percentage_2pt' in the 'player_and1_attempts_minimized' DataFrame.

41. `player_and1_attempts_minimized['And_1_Percentage_3pt'] = player_and1_attempts['And_1s_3pt'] / player_and1_attempts['Shooting_Fouls_3pt']`
Calculates the 3pt and-1 percentage by dividing the 'And_1s_3pt' column of the 'player_and1_attempts' DataFrame by the 'Shooting_Fouls_3pt' column. The result is assigned to a new column called 'And_1_Percentage_3pt' in the 'player_and1_attempts_minimized' DataFrame.

42. `player_and1_attempts_minimized`
Displays the 'player_and1_attempts_minimized' DataFrame, which now includes the calculated and-1 percentage columns.

43. `all_time_total_and1_pct_leaders = player_and1_attempts_minimized.sort_values(by=['And_1_Percentage' 'Shooting_Fouls'], ascending=[False, True]).head(20)`
Calculates the total and-1 percentage leaders all time by sorting the 'player_and1_attempts_minimized' DataFrame based on the 'And_1_Percentage' and 'Shooting_Fouls' columns in descending and ascending order, respectively. The top 20 rows are selected and assigned to the variable 'all_time_total_and1_pct_leaders'.

44. `all_time_total_and1_pct_leaders['Player Name'] = all_time_total_and1_pct_leaders['Player_ID'].apply`
Adds a new column 'Player Name' to the 'all_time_total_and1_pct_leaders' DataFrame by applying the 'players.find_player_by_id' function to the 'Player_ID' column and extracting the 'full_name' property.

45. `all_time_total_and1_pct_leaders[['Player Name', 'And_1s', 'Shooting_Fouls', 'And_1_Percentage']]`
Selects specific columns ('Player Name', 'And_1s', 'Shooting_Fouls', 'And_1_Percentage') from the 'all_time_total_and1_pct_leaders' DataFrame.

46. `all_time_lowest_total_and1_pct = player_and1_attempts_minimized.sort_values(by=['And_1_Percentage', 'Shooting_Fouls'], ascending=[True, True]).head(20)`
Calculates the worst and-1 percentage all time by sorting the 'player_and1_attempts_minimized' DataFrame based on the 'And_1_Percentage' and 'Shooting_Fouls' columns in ascending order. The top 20 rows are selected and assigned to the variable 'all_time_lowest_total_and1_pct'.

47. `all_time_lowest_total_and1_pct['Player Name'] = all_time_lowest_total_and1_pct['Player_ID'].apply(p`
Adds a new column 'Player Name' to the 'all_time_lowest_total_and1_pct' DataFrame by applying the 'players.find_player_by_id' function to the 'Player_ID' column and extracting the 'full_name' property.

48. `all_time_lowest_total_and1_pct[['Player Name', 'And_1s', 'Shooting_Fouls', 'And_1_Percentage']]`
Selects specific columns ('Player Name', 'And_1s', 'Shooting_Fouls', 'And_1_Percentage') from the 'all_time_lowest_total_and1_pct' DataFrame.

49. `all_time_2pt_and1_pct_leaders = player_and1_attempts_minimized.sort_values(by=['And_1_Percentage_2p 'Shooting_Fouls'], ascending=[False, True]).head(20)`
Calculates the 2pt and-1 percentage leaders all time by sorting the 'player_and1_attempts_minimized' DataFrame based on the 'And_1_Percentage_2pt' and 'Shooting_Fouls' columns in descending and ascending order, respectively. The top 20 rows are selected and assigned to the variable 'all_time_2pt_and1_pct_leaders'.

50. `all_time_2pt_and1_pct_leaders['Player Name'] = all_time_2pt_and1_pct_leaders['Player_ID'].apply(pla`
Adds a new column 'Player Name' to the 'all_time_2pt_and1_pct_leaders' DataFrame by applying the 'players.find_player_by_id' function to the 'Player_ID' column and extracting the 'full_name' property.

51. `all_time_2pt_and1_pct_leaders[['Player Name', 'And_1s', 'Shooting_Fouls', 'And_1_Percentage']]`
Selects specific columns ('Player Name', 'And_1s', 'Shooting_Fouls', 'And_1_Percentage') from the 'all_time_2pt_and1_pct_leaders' DataFrame.

52. `all_time_lowest_2pt_and1_pct = player_and1_attempts_minimized.sort_values(by=['And_1_Percentage_2pt 'Shooting_Fouls'], ascending=[True, True]).head(20)`

Calculates the worst 2pt and-1 percentage all time by sorting the 'player_and1_attempts_minimized' DataFrame based on the 'And_1_Percentage_2pt' and 'Shooting_Fouls' columns in ascending order. The top 20 rows are selected and assigned to the variable 'all_time_lowest_2pt_and1_pct'.

53. `all_time_lowest_2pt_and1_pct['Player Name'] = all_time_lowest_2pt_and1_pct['Player_ID'].apply(playe`
Adds a new column 'Player Name' to the 'all_time_lowest_2pt_and1_pct' DataFrame by applying the 'players.find_player_by_id' function to the 'Player_ID' column and extracting the 'full_name' property.

54. `all_time_lowest_2pt_and1_pct[['Player Name', 'And_1s', 'Shooting_Fouls', 'And_1_Percentage']]`
Selects specific columns ('Player Name', 'And_1s', 'Shooting_Fouls', 'And_1_Percentage') from the 'all_time_lowest_2pt_and1_pct' DataFrame.

55. `mask = player_and1_attempts['And_1s_3pt'] >= 3`
Creates a boolean mask by checking if the 'And_1s_3pt' column in the 'player_and1_attempts' DataFrame is greater than or equal to 3.

56. `player_and1_attempts_minimized = player_and1_attempts.loc[mask]`
Filters the 'player_and1_attempts' DataFrame by using the boolean mask and assigns the filtered DataFrame to the variable 'player_and1_attempts_minimized'.

57. `player_and1_attempts_minimized['And_1_Percentage'] = player_and1_attempts['And_1s']`
`/ player_and1_attempts['Shooting_Fouls']`
Calculates the and-1 percentage by dividing the 'And_1s' column in the 'player_and1_attempts' DataFrame by the 'Shooting_Fouls' column. The result is assigned to the 'And_1_Percentage' column in the 'player_and1_attempts_minimized' DataFrame.

58. `player_and1_attempts_minimized['And_1_Percentage_2pt'] = player_and1_attempts['And_1s_2pt']`
`/ player_and1_attempts['Shooting_Fouls_2pt']`
Calculates the 2pt and-1 percentage by dividing the 'And_1s_2pt' column in the 'player_and1_attempts' DataFrame by the 'Shooting_Fouls_2pt' column. The result is assigned to the 'And_1_Percentage_2pt' column in the 'player_and1_attempts_minimized' DataFrame.

59. `player_and1_attempts_minimized['And_1_Percentage_3pt'] = player_and1_attempts['And_1s_3pt']`
`/ player_and1_attempts['Shooting_Fouls_3pt']`
Calculates the 3pt and-1 percentage by dividing the 'And_1s_3pt' column in the 'player_and1_attempts' DataFrame by the 'Shooting_Fouls_3pt' column. The result is assigned to the 'And_1_Percentage_3pt' column in the 'player_and1_attempts_minimized' DataFrame.

60. `player_and1_attempts_minimized`
Returns the 'player_and1_attempts_minimized' DataFrame.

61. `all_time_3pt_and1_pct_leaders = player_and1_attempts_minimized.sort_values(by=['And_1_Percentage_3p`
`'Shooting_Fouls'], ascending=[False, True]).head(20)`
Sorts the 'player_and1_attempts_minimized' DataFrame based on the 'And_1_Percentage_3pt' and 'Shooting_Fouls' columns in descending and ascending order, respectively. The top 20 rows are selected and assigned to the 'all_time_3pt_and1_pct_leaders' DataFrame.

62. `all_time_3pt_and1_pct_leaders['Player Name'] = all_time_3pt_and1_pct_leaders['Player_ID'].apply(pla`
Adds a new column 'Player Name' to the 'all_time_3pt_and1_pct_leaders' DataFrame by applying the 'players.find_player_by_id' function to the 'Player_ID' column and extracting the 'full_name' property.

63. `all_time_3pt_and1_pct_leaders[['Player Name', 'And_1s', 'Shooting_Fouls', 'And_1_Percentage']]`
Selects specific columns ('Player Name', 'And_1s', 'Shooting_Fouls', 'And_1_Percentage') from the 'all_time_3pt_and1_pct_leaders' DataFrame.

65. `all_time_lowest_3pt_and1_pct = player_and1_attempts_minimized.sort_values(by=['And_1_Percentage_3pt`
`'Shooting_Fouls'], ascending=[True, True]).head(20)`
Sorts the 'player_and1_attempts_minimized' DataFrame based on the 'And_1_Percentage_3pt' and 'Shooting_Fouls' columns in ascending order. The top 20 rows with the lowest percentages are selected and assigned to the 'all_time_lowest_3pt_and1_pct' DataFrame.

66. `all_time_lowest_3pt_and1_pct['Player Name'] = all_time_lowest_3pt_and1_pct['Player_ID'].apply(playe`
Adds a new column 'Player Name' to the 'all_time_lowest_3pt_and1_pct' DataFrame by applying the 'players.find_player_by_id' function to the 'Player_ID' column and extracting the 'full_name' property.

67. `all_time_lowest_3pt_and1_pct[['Player Name', 'And_1s', 'Shooting_Fouls', 'And_1_Percentage']]`
Selects specific columns ('Player Name', 'And_1s', 'Shooting_Fouls', 'And_1_Percentage') from the 'all_time_lowest_3pt_and1_pct' DataFrame.

68. `team_and1_attempts = and1_df.groupby(['Team']).sum()`
Groups all the and-1 plays by team using the groupby function and calculates the sum of each team's and-1 statistics. The result is stored in the team_and1_attempts DataFrame.

69. `team_and1_attempts.reset_index(inplace=True)`
Resets the index of the team_and1_attempts DataFrame after the grouping operation.

70. `team_and1_attempts`
Displays the team_and1_attempts DataFrame, which contains the team names and the summed and-1 statistics for each team.

71. `team_all_time_and1_leaders_total = team_and1_attempts.sort_values(by=['And_1s', 'Shooting_Fouls'], ascending=[False, True])`
Sorts the team_and1_attempts DataFrame based on the 'And_1s' and 'Shooting_Fouls' columns in descending and ascending order respectively. This will give us the teams with the highest total and-1 plays. The result is assigned to the team_all_time_and1_leaders_total DataFrame.

72. `team_all_time_and1_leaders_total`
Displays the team_all_time_and1_leaders_total DataFrame, which contains the teams ranked by the highest total number of and-1 plays.

73. `team_all_time_and1_leaders_2pt = team_and1_attempts.sort_values(by=['And_1s_2pt', 'Shooting_Fouls'] ascending=[False, True])`
Sorts the team_and1_attempts DataFrame based on the 'And_1s_2pt' and 'Shooting_Fouls' columns in descending and ascending order, respectively. This will give us the teams with the highest total and-1 plays for 2-point shots. The result is assigned to the team_all_time_and1_leaders_2pt DataFrame.

74. `team_all_time_and1_leaders_2pt`
Displays the team_all_time_and1_leaders_2pt DataFrame, which contains the teams ranked by the highest total number of and-1 plays for 2-point shots.

75. `team_all_time_and1_leaders_3pt = team_and1_attempts.sort_values(by=['And_1s_3pt', 'Shooting_Fouls'] ascending=[False, True])`
Sorts the team_and1_attempts DataFrame based on the 'And_1s_3pt' and 'Shooting_Fouls' columns in descending and ascending order, respectively. This will give us the teams with the highest total and-1 plays for 3-point shots. The result is assigned to the team_all_time_and1_leaders_3pt DataFrame.

76. `team_all_time_and1_leaders_3pt`
Displays the team_all_time_and1_leaders_3pt DataFrame, which contains the teams ranked by the highest total number of and-1 plays for 3-point shots.

77. `team_and1_attempts_by_season = and1_df.groupby(['Team', 'Season']).sum()`
Calculates the total and-1 attempts for each team and season by grouping the and1_df DataFrame by the 'Team' and 'Season' columns and summing the values.

78. `team_and1_attempts_by_season.reset_index(inplace=True)`
Resets the index of the team_and1_attempts_by_season DataFrame in place, so that the 'Team' and 'Season' columns become regular columns instead of part of the index.

79. `team_and1_attempts_by_season`
Displays the team_and1_attempts_by_season DataFrame, which contains the total and-1 attempts for each team and season.

80. `team_season_and1_leaders_total = team_and1_attempts_by_season.sort_values(by=['And_1s', 'Shooting_Fouls'], ascending=[False, True]).head(50)`
Sorts the team_and1_attempts_by_season DataFrame based on the 'And_1s' and 'Shooting_Fouls' columns in descending and ascending order, respectively. This will give us the teams with the highest total and-1 plays across all seasons. The top 50 teams are selected using the head() function and assigned to the team_season_and1_leaders_total DataFrame.

81. `team_season_and1_leaders_total`
Displays the team_season_and1_leaders_total DataFrame, which contains the top 50 teams ranked by the highest total number of and-1 plays across all seasons.

82. `team_season_and1_lowest_total = team_and1_attempts_by_season.sort_values(by=['And_1s', 'Shooting_Fouls'], ascending=[True, True]).head(50)`
Sorts the team_and1_attempts_by_season DataFrame based on the 'And_1s' and 'Shooting_Fouls' columns in ascending order for both columns. This will give us the teams with the lowest total and-1 plays across all seasons. The top 50 teams with the lowest values are selected using the head() function and assigned to the team_season_and1_lowest_total DataFrame.

83. `team_season_and1_lowest_total`
Displays the team_season_and1_lowest_total DataFrame, which contains the top 50 teams ranked by the lowest total number of and-1 plays across all seasons.

84. `team_season_and1_leaders_2pt = team_and1_attempts_by_season.sort_values(by=['And_1s_2pt', 'Shooting_Fouls'], ascending=[False, True]).head(50)`
Sorts the team_and1_attempts_by_season DataFrame based on the 'And_1s_2pt' and 'Shooting_Fouls' columns in descending and ascending order, respectively. This will give us the teams with the highest total and-1 plays for 2-point shots across all seasons. The top 50 teams are selected using the head() function and assigned to the team_season_and1_leaders_2pt DataFrame.

85. `team_season_and1_leaders_2pt`
Displays the team_season_and1_leaders_2pt DataFrame, which contains the top 50 teams ranked by the highest total number of and-1 plays for 2-point shots across all seasons.

86. `team_season_and1_lowest_2pt = team_and1_attempts_by_season.sort_values(by=['And_1s_2pt', 'Shooting_Fouls'], ascending=[True, True]).head(50)`
Sorts the team_and1_attempts_by_season DataFrame based on the 'And_1s_2pt' and 'Shooting_Fouls' columns in ascending order for both columns. This will give us the teams with the lowest total and-1 plays for 2-point shots across all seasons. The top 50 teams with the lowest values are selected using the head() function and assigned to the team_season_and1_lowest_2pt DataFrame.

87. `team_season_and1_lowest_2pt`
Displays the team_season_and1_lowest_2pt DataFrame, which contains the top 50 teams ranked by the lowest total number of and-1 plays for 2-point shots across all seasons.

88. `team_season_and1_leaders_3pt = team_and1_attempts_by_season.sort_values(by=['And_1s_3pt', 'Shooting_Fouls'], ascending=[False, True]).head(50)`
Sorts the team_and1_attempts_by_season DataFrame based on the 'And_1s_3pt' and 'Shooting_Fouls' columns in descending and ascending order, respectively. This will give us the teams with the highest total and-1 plays for 3-point shots across all seasons. The top 50 teams are selected using the head() function and assigned to the team_season_and1_leaders_3pt DataFrame.

89. `team_season_and1_leaders_3pt`
Displays the team_season_and1_leaders_3pt DataFrame, which contains the top 50 teams ranked by the highest total number of and-1 plays for 3-point shots across all seasons.

90. `team_season_and1_lowest_3pt = team_and1_attempts_by_season.sort_values(by=['And_1s_3pt', 'Shooting_Fouls'], ascending=[True, True]).head(50)`
Sorts the team_and1_attempts_by_season DataFrame based on the 'And_1s_3pt' and 'Shooting_Fouls' columns in ascending order for both columns. This will give us the teams with the lowest total and-1 plays for 3-point shots across all seasons. The top 50 teams with the lowest values are selected using the head() function and assigned to the team_season_and1_lowest_3pt DataFrame.

91. `team_season_and1_lowest_3pt`
Displays the team_season_and1_lowest_3pt DataFrame, which contains the top 50 teams ranked by the lowest total number of and-1 plays for 3-point shots across all seasons.

92. `season_and1_attempts = and1_df.groupby(['Season']).sum()`
Calculates the sum of and-1 attempts for each season by grouping the data in the and1_df DataFrame based on the 'Season' column. The result is assigned to the season_and1_attempts DataFrame.

93. `season_and1_attempts.reset_index(inplace=True)`
Resets the index of the season_and1_attempts DataFrame to ensure a continuous index after the groupby operation. The changes are made in place by setting the inplace parameter to True.

94. `season_and1_attempts`
Displays the season_and1_attempts DataFrame, which contains the sum of and-1 attempts for each season.

95. `def numTeamsInTheLeague(season):`
Defines a function named numTeamsInTheLeague that takes a season as an input parameter.

96. `if season == "1996-97" or season == "1997-98" or season == "1998-99" or season == "1999-00" or season == "2000-01" or season == "2001-02" or season == "2002-03" or season == "2003-04":`
Checks if the value of the season parameter matches any of the specified seasons from 1996-97 to 2003-04.

97. `return 29`
Returns the value 29 if the condition in line 96 is true.

98. `else:`
Specifies the else block when the condition in line 96 is false.

99. `return 30`
Returns the value 30 if the condition in line 98 is true.

100. `season_and1_attempts['Num_Teams'] = season_and1_attempts['Season'].apply(numTeamsInTheLeague)`
Applies the number of teams in the league to each season by using the apply function on the 'Season' column of the season_and1_attempts DataFrame and passing the numTeamsInTheLeague function as the argument. The result is assigned to the 'Num_Teams' column.

101. `season_and1_attempts['Avg_And_1s'] = (season_and1_attempts['And_1s'] / season_and1_attempts['Num_T`
Applies the average number of and-1 plays per team to each season by dividing the 'And_1s' column of the season_and1_attempts DataFrame by the 'Num_Teams' column and rounding the result to 2 decimal places. The result is assigned to the 'Avg_And_1s' column.

102. `season_avg_and1s_dict = dict()`
Initializes an empty dictionary named season_avg_and1s_dict.

103. `for index, row in season_and1_attempts.iterrows():`
Iterates over the rows of the season_and1_attempts DataFrame using the iterrows() function.

104. `season_avg_and1s_dict[row['Season']] = row['Avg_And_1s']`
Assigns the value of the 'Avg_And_1s' column from each row of the DataFrame to the corresponding season as a key in the season_avg_and1s_dict dictionary.

105. `team_and1_attempts_by_season['And_1+'] = (team_and1_attempts_by_season['And_1s'] / team_and1_attempts_by_season['Season'].apply(lambda x: season_avg_and1s_dict[x]) * 100).round(2)`
Calculates the 'And_1+' column of the team_and1_attempts_by_season DataFrame by dividing the 'And_1s' column by the average and-1 plays per team for each season, multiplied by 100. The result is rounded to 2 decimal places.

106. `team_season_and1_leaders_total = team_and1_attempts_by_season.sort_values(by=['And_1+', 'Shooting_Fouls'], ascending=[False, True]).head(50)`
Sorts the `team_and1_attempts_by_season` DataFrame based on the `'And_1+'` and `'Shooting_Fouls'` columns in descending and ascending order respectively. This will give us the teams with the highest total percentage of and-1 plays. The result is assigned to the `team_season_and1_leaders_total` DataFrame.

107. `team_season_and1_leaders_total`
Displays the `team_season_and1_leaders_total` DataFrame, which contains the top 50 teams with the highest total percentage of and-1 plays.

108. `team_season_and1_lowest_total = team_and1_attempts_by_season.sort_values(by=['And_1+', 'Shooting_Fouls'], ascending=[True, True]).head(50)`

Sorts the `team_and1_attempts_by_season` DataFrame based on the `'And_1+'` and `'Shooting_Fouls'` columns in ascending order for both columns. This will give us the teams with the lowest total percentage of and-1 plays. The result is assigned to the `team_season_and1_lowest_total` DataFrame.

`109. team_season_and1_lowest_total`

Displays the `team_season_and1_lowest_total` DataFrame, which contains the top 50 teams with the lowest total percentage of and-1 plays.

`111. all_teams = teams.get_teams()`

Retrieves information about all NBA teams using the get_teams() method from the teams module. The result is assigned to the all_teams variable.

`112. team_ids = ['id': team['id'], 'name': team['abbreviation'] for team in all_teams]`

Creates a list of dictionaries containing the team ID and abbreviation for each team in the all_teams list. The list comprehension iterates over all_teams and creates a dictionary for each team with keys 'id' and 'name' representing the team ID and abbreviation, respectively. The resulting list is assigned to the team_ids variable.

`114. team_ids_dict = dict()`

Creates an empty dictionary to store the mapping between team abbreviations and team IDs. The dictionary will be populated in the following loop.

`115. possible_team_abbrevs = team_and1_attempts_by_season['Team'].unique()`

Retrieves the unique team abbreviations from the team_and1_attempts_by_season DataFrame and assigns them to the possible_team_abbrevs variable.

`116. current_team_abbrevs = [team['abbreviation'] for team in all_teams]`

Creates a list of team abbreviations by iterating over the all_teams list of dictionaries and extracting the 'abbreviation' value for each team. The resulting list is assigned to the current_team_abbrevs variable.

`118. for value in possible_team_abbrevs:`

Iterates over each value in the possible_team_abbrevs list.

`119. if value in current_team_abbrevs:`

Checks if the current value is present in the current_team_abbrevs list.

`120. team_ids_dict[value] = teams.find_team_by_abbreviation(value)['id']`

If the value is found in current_team_abbrevs, it retrieves the team ID using the find_team_by_abbreviation() function from the teams module and assigns it to the corresponding key in the team_ids_dict dictionary.

`123. else:`

If the value is not found in current_team_abbrevs, it accounts for teams that have changed location.

`124. if value == 'CHH':`

Checks if the value is 'CHH'.

`125. team_ids_dict[value] = teams.find_team_by_abbreviation('CHA')['id']`

If the value is 'CHH', it retrieves the team ID for 'CHA' and assigns it to the corresponding key in the team_ids_dict dictionary.

`126. elif value == 'NJN':`

Checks if the value is 'NJN'.

`127. team_ids_dict[value] = teams.find_team_by_abbreviation('BKN')['id']`

If the value is 'NJN', it retrieves the team ID for 'BKN' and assigns it to the corresponding key in the team_ids_dict dictionary.

`128. elif value == 'NOH':`

Checks if the value is 'NOH'.

`129. team_ids_dict[value] = teams.find_team_by_abbreviation('NOP')['id']`

If the value is 'NOH', it retrieves the team ID for 'NOP' and assigns it to the corresponding key in the team_ids_dict dictionary.

`130. elif value == 'NOK':`

Checks if the value is 'NOK'.

131. `team_ids_dict[value] = teams.find_team_by_abbreviation('NOP')['id']`
If the value is 'NOK', it retrieves the team ID for 'NOP' and assigns it to the corresponding key in the team_ids_dict dictionary.

132. `elif value == 'SEA':`
Checks if the value is 'SEA'.

133. `team_ids_dict[value] = teams.find_team_by_abbreviation('OKC')['id']`
If the value is 'SEA', it retrieves the team ID for 'OKC' and assigns it to the corresponding key in the team_ids_dict dictionary.

134. `elif value == 'VAN':`
Checks if the value is 'VAN'.

135. `team_ids_dict[value] = teams.find_team_by_abbreviation('MEM')['id']`
If the value is 'VAN', it retrieves the team ID for 'MEM' and assigns it to the corresponding key in the team_ids_dict dictionary.

136. `team_and1_attempts_by_season['Team_ID'] = (team_and1_attempts_by_season['Team'].apply(lambda x: team_ids_dict[x]))`
Assigns the corresponding team ID to each team in the team_and1_attempts_by_season DataFrame using the team_ids_dict dictionary. The lambda function is applied to the 'Team' column, mapping each team name to its corresponding ID in the dictionary. The resulting IDs are then assigned to the 'Team_ID' column.

137. `team_and1_attempts_by_season`
Outputs the updated team_and1_attempts_by_season DataFrame, which now includes the 'Team_ID' column with the corresponding team IDs for each team.

138. `file_path = 'team_yearly_stats.csv'`
Assigns the file path of the team_yearly_stats.csv file to the variable file_path. This file contains yearly statistics for NBA teams.

139. `nba_yearly_stats = pd.read_csv(file_path)`
Reads the CSV file specified by the file_path using the pandas read_csv() function and assigns the resulting DataFrame to the variable nba_yearly_stats.

140. `for index, row in team_and1_attempts_by_season.iterrows():`
Iterates over the rows of the team_and1_attempts_by_season DataFrame, assigning the index and row values to the variables index and row, respectively.

141. `team = team_ids_dict[row['Team']]`
Retrieves the team ID from the team_ids_dict dictionary based on the value in the 'Team' column of the current row and assigns it to the variable team.

142. `season = row['Season']`
Retrieves the season value from the 'Season' column of the current row and assigns it to the variable season.

143. `yearly_row = nba_yearly_stats.loc[(nba_yearly_stats['TEAM_ID'] == team) & (nba_yearly_stats['YEAR == season)]`
Filters the nba_yearly_stats DataFrame to retrieve the row where the 'TEAM_ID' column matches the team ID and the 'YEAR' column matches the season. The resulting row is assigned to the variable yearly_row.

144. `team_and1_attempts_by_season.at[index, 'PPG'] = yearly_row['PTS'].values[0]`
Updates the 'PPG' column of the team_and1_attempts_by_season DataFrame at the current index with the value from the 'PTS' column of the yearly_row DataFrame.

145. `team_and1_attempts_by_season.at[index, 'Pts_Rank'] = yearly_row['PTS_RANK'].values[0]`
Updates the 'Pts_Rank' column of the team_and1_attempts_by_season DataFrame at the current index with the value from the 'PTS_RANK' column of the yearly_row DataFrame.

146. `team_and1_attempts_by_season.at[index, 'Win_Pct'] = yearly_row['WIN_PCT'].values[0]`
Updates the 'Win_Pct' column of the team_and1_attempts_by_season DataFrame at the current index with the value from the 'WIN_PCT' column of the yearly_row DataFrame.

147. `team_and1_attempts_by_season.at[index, 'FTA_Per_Game'] = yearly_row['FTA'].values[0]`
Updates the 'FTA_Per_Game' column of the team_and1_attempts_by_season DataFrame at the current index with the value from the 'FTA' column of the yearly_row DataFrame.

148. `and1_plus = team_and1_attempts_by_season['And_1+']`
Retrieves the 'And_1+' column from the team_and1_attempts_by_season DataFrame and assigns it to the variable and1_plus.

149. `win_pct = team_and1_attempts_by_season['Win_Pct']`
Retrieves the 'Win_Pct' column from the team_and1_attempts_by_season DataFrame and assigns it to the variable win_pct.

150. `plt.scatter(and1_plus, win_pct)`
Creates a scatter plot using the and1_plus values as the x-axis and the win_pct values as the y-axis.

151. `plt.xlabel('And1+')`
Sets the label for the x-axis of the scatter plot to 'And1+'.

152. `plt.ylabel('Win %')`
Sets the label for the y-axis of the scatter plot to 'Win %'.

153. `plt.show()`
Displays the scatter plot.

154. `slope, intercept, r_value, p_value, std_err = stats.linregress(and1_plus, win_pct)`
Performs linear regression analysis using the and1_plus values as the independent variable and the win_pct values as the dependent variable. The resulting slope, intercept, correlation coefficient (r_value), p-value (p_value), and standard error (std_err) are assigned to their respective variables.

155. `print("Slope:", slope)`
Prints the slope of the linear regression line.

156. `print("Intercept:", intercept)`
Prints the intercept of the linear regression line.

157. `print("R-squared:", r_value**2)`
Prints the coefficient of determination (R-squared) of the linear regression.

158. `print("p-value:", p_value)`
Prints the p-value associated with the null hypothesis in the linear regression.

159. `print("Standard Error:", std_err)`
Prints the standard error of the estimated slope and intercept in the linear regression.

160. `and1_plus = team_and1_attempts_by_season['And_1+']`
Retrieves the 'And_1+' column from the team_and1_attempts_by_season DataFrame and assigns it to the variable and1_plus.

161. `pts_rank = team_and1_attempts_by_season['Pts_Rank']`
Retrieves the 'Pts_Rank' column from the team_and1_attempts_by_season DataFrame and assigns it to the variable pts_rank.

162. `plt.scatter(and1_plus, pts_rank)`
Creates a scatter plot using the and1_plus values as the x-axis and the pts_rank values as the y-axis.

163. `plt.xlabel('And1+')`
Sets the label for the x-axis of the scatter plot to 'And1+'.

164. `plt.ylabel('Point Rank')`
Sets the label for the y-axis of the scatter plot to 'Point Rank'.

165. `plt.show()`

Displays the scatter plot.

`166. slope, intercept, r_value, p_value, std_err = stats.linregress(and1_plus, pts_rank)`
Performs linear regression analysis using the and1_plus values as the independent variable and the pts_rank values as the dependent variable. The resulting slope, intercept, correlation coefficient (r_value), p-value (p_value), and standard error (std_err) are assigned to their respective variables.

`167. print("Slope:", slope)`
Prints the slope of the linear regression line.

`168. print("Intercept:", intercept)`
Prints the intercept of the linear regression line.

`169. print("R-squared:", r_value**2)`
Prints the coefficient of determination (R-squared) of the linear regression.

`170. print("p-value:", p_value)`
Prints the p-value associated with the null hypothesis in the linear regression.

`171. print("Standard Error:", std_err)`
Prints the standard error of the estimated slope and intercept in the linear regression.

`172. and1_plus = team_and1_attempts_by_season['And_1+']`
Retrieves the 'And_1+' column from the team_and1_attempts_by_season DataFrame and assigns it to the variable and1_plus.

`173. ppg = team_and1_attempts_by_season['PPG']`
Retrieves the 'PPG' column from the team_and1_attempts_by_season DataFrame and assigns it to the variable ppg.

`174. plt.scatter(and1_plus, ppg)`
Creates a scatter plot using the and1_plus values as the x-axis and the ppg values as the y-axis.

`175. plt.xlabel('And1+')`
Sets the label for the x-axis of the scatter plot to 'And1+'.

`176. plt.ylabel('Point Per Game')`
Sets the label for the y-axis of the scatter plot to 'Point Per Game'.

`177. plt.show()`
Displays the scatter plot.

`178. slope, intercept, r_value, p_value, std_err = stats.linregress(and1_plus, ppg)`
Performs linear regression analysis using the and1_plus values as the independent variable and the ppg values as the dependent variable. The resulting slope, intercept, correlation coefficient (r_value), p-value (p_value), and standard error (std_err) are assigned to their respective variables.

`179. print("Slope:", slope)`
Prints the slope of the linear regression line.

`180. print("Intercept:", intercept)`
Prints the intercept of the linear regression line.

`181. print("R-squared:", r_value**2)`
Prints the coefficient of determination (R-squared) of the linear regression.

`182. print("p-value:", p_value)`
Prints the p-value associated with the null hypothesis in the linear regression.

`183. print("Standard Error:", std_err)`
Prints the standard error of the estimated slope and intercept in the linear regression.

`184. and1_plus = team_and1_attempts_by_season['And_1+']`
Retrieves the 'And_1+' column from the team_and1_attempts_by_season DataFrame and assigns it to the variable and1_plus.

`185. fta = team_and1_attempts_by_season['FTA_Per_Game']`

Retrieves the 'FTA_Per_Game' column from the team_and1_attempts_by_season DataFrame and assigns it to the variable fta.

186. `plt.scatter(and1_plus, fta)`
Creates a scatter plot using the and1_plus values as the x-axis and the fta values as the y-axis.

187. `plt.xlabel('And1+')`
Sets the label for the x-axis of the scatter plot to 'And1+'.

188. `plt.ylabel('Free Throws Attempted/Game')`
Sets the label for the y-axis of the scatter plot to 'Free Throws Attempted/Game'.

189. `plt.show()`
Displays the scatter plot.

190. `slope, intercept, r_value, p_value, std_err = stats.linregress(and1_plus, fta)`
Performs linear regression analysis using the and1_plus values as the independent variable and the fta values as the dependent variable. The resulting slope, intercept, correlation coefficient (r_value), p-value (p_value), and standard error (std_err) are assigned to their respective variables.

191. `print("Slope:", slope)`
Prints the slope of the linear regression line.

192. `print("Intercept:", intercept)`
Prints the intercept of the linear regression line.

193. `print("R-squared:", r_value**2)`
Prints the coefficient of determination (R-squared) of the linear regression.

194. `print("p-value:", p_value)`
Prints the p-value associated with the null hypothesis in the linear regression.

195. `print("Standard Error:", std_err)`
Prints the standard error of the estimated slope and intercept in the linear regression.

196. `team_and1_commits = and1_df.groupby(['Opposing_Team']).sum()`
Groups the data in the and1_df DataFrame by the 'Opposing_Team' column and calculates the sum of each group. The result is assigned to the team_and1_commits DataFrame, which represents the total number of and-1 situations committed by each opposing team.

197. `team_and1_commits.reset_index(inplace=True)`
Resets the index of the team_and1_commits DataFrame. This operation modifies the DataFrame in place, updating the index to a default numeric index.

198. `team_and1_commits`
Outputs the updated team_and1_commits DataFrame, which now contains the total number of and-1 situations committed by each opposing team, with the index reset.

199. `team_all_time_and1_committers_total = team_and1_commits.sort_values(by=['And_1s', 'Shooting_Fouls' ascending=[False, True])`
Sorts the team_and1_commits DataFrame by the 'And_1s' column in descending order and then by the 'Shooting_Fouls' column in ascending order. The result is assigned to the team_all_time_and1_committers_total DataFrame, which represents the total number of and-1 situations committed by each opposing team, sorted in descending order by the number of and-1s and ascending order by the number of shooting fouls.

200. `team_all_time_and1_committers_total`
Outputs the updated team_all_time_and1_committers_total DataFrame, which now contains the total number of and-1 situations committed by each opposing team, sorted in descending order by the number of and-1s and ascending order by the number of shooting fouls.

201. `team_all_time_and1_committers_2pt = team_and1_commits.sort_values(by=['And_1s_2pt', 'Shooting_Fouls'], ascending=[False, True])`
Sorts the team_and1_commits DataFrame by the 'And_1s_2pt' column in descending order and then by the 'Shooting_Fouls' column in ascending order. The result is assigned to the team_all_time_and1_committers_2pt

DataFrame, which represents the total number of and-1 situations committed by each opposing team in 2-point shooting situations, sorted in descending order by the number of 2-point and-1s and ascending order by the number of shooting fouls.

202. `team_all_time_and1_committers_2pt`
Outputs the updated team_all_time_and1_committers_2pt DataFrame, which now contains the total number of and-1 situations committed by each opposing team in 2-point shooting situations, sorted in descending order by the number of 2-point and-1s and ascending order by the number of shooting fouls.

203. `team_all_time_and1_committers_3pt = team_and1_commits.sort_values(by=['And_1s_3pt', 'Shooting_Fouls'], ascending=[False, True])`
Sorts the team_and1_commits DataFrame by the 'And_1s_3pt' column in descending order and then by the 'Shooting_Fouls' column in ascending order. The result is assigned to the team_all_time_and1_committers_3pt DataFrame, which represents the total number of and-1 situations committed by each opposing team in 3-point shooting situations, sorted in descending order by the number of 3-point and-1s and ascending order by the number of shooting fouls.

204. `team_all_time_and1_committers_3pt`
Outputs the updated team_all_time_and1_committers_3pt DataFrame, which now contains the total number of and-1 situations committed by each opposing team in 3-point shooting situations, sorted in descending order by the number of 3-point and-1s and ascending order by the number of shooting fouls.

205. `team_and1_commits_by_season = and1_df.groupby(['Opposing_Team', 'Season']).sum()`
Groups the and1_df DataFrame by both the 'Opposing_Team' and 'Season' columns and calculates the sum of each group. The result is assigned to the team_and1_commits_by_season DataFrame, which represents the total number of and-1 situations committed by each opposing team in each season.

206. `team_and1_commits_by_season.reset_index(inplace=True)`
Resets the index of the team_and1_commits_by_season DataFrame. This operation converts the grouped columns ('Opposing_Team' and 'Season') into regular columns and assigns a default index to the DataFrame.

207. `team_and1_commits_by_season`
Outputs the updated team_and1_commits_by_season DataFrame, which now contains the total number of and-1 situations committed by each opposing team in each season.

208. `team_season_and1_highest_committers_total = team_and1_commits_by_season.sort_values(by=['And_1s', 'Shooting_Fouls'], ascending=[False, True]).head(50)`
Sorts the team_and1_commits_by_season DataFrame by the 'And_1s' and 'Shooting_Fouls' columns in descending and ascending order respectively. The resulting DataFrame contains the top 50 rows, representing the opposing teams and seasons with the highest number of and-1 situations committed. This DataFrame is assigned to the team_season_and1_highest_committers_total variable.

209. `team_season_and1_highest_committers_total`
Outputs the team_season_and1_highest_committers_total DataFrame, which contains the top 50 rows representing the opposing teams and seasons with the highest number of and-1 situations committed.

210. `team_season_and1_lowest_committed_total = team_and1_commits_by_season.sort_values(by=['And_1s', 'Shooting_Fouls'], ascending=[True, True]).head(50)`
Sorts the team_and1_commits_by_season DataFrame by the 'And_1s' and 'Shooting_Fouls' columns in ascending order for both. The resulting DataFrame contains the bottom 50 rows, representing the opposing teams and seasons with the lowest number of and-1 situations committed. This DataFrame is assigned to the team_season_and1_lowest_committed_total variable.

211. `team_season_and1_lowest_committed_total`
Outputs the team_season_and1_lowest_committed_total DataFrame, which contains the bottom 50 rows representing the opposing teams and seasons with the lowest number of and-1 situations committed.

212. `team_season_and1_highest_committers_2pt = team_and1_commits_by_season.sort_values(by=['And_1s_2pt' 'Shooting_Fouls'], ascending=[False, True]).head(50)`

Sorts the team_and1_commits_by_season DataFrame by the 'And_1s_2pt' and 'Shooting_Fouls' columns in descending order for 'And_1s_2pt' and ascending order for 'Shooting_Fouls'. The resulting DataFrame contains the top 50 rows, representing the opposing teams and seasons with the highest number of and-1 situations committed in two-point plays. This DataFrame is assigned to the team_season_and1_highest_committers_2pt variable.

213. `team_season_and1_highest_committers_2pt`
Outputs the team_season_and1_highest_committers_2pt DataFrame, which contains the top 50 rows representing the opposing teams and seasons with the highest number of and-1 situations committed in two-point plays.

214. `team_season_and1_lowest_committers_2pt = team_and1_commits_by_season.sort_values(by=['And_1s_2pt', 'Shooting_Fouls'], ascending=[True, True]).head(50)`
Sorts the team_and1_commits_by_season DataFrame by the 'And_1s_2pt' and 'Shooting_Fouls' columns in ascending order for both columns. The resulting DataFrame contains the top 50 rows, representing the opposing teams and seasons with the lowest number of and-1 situations committed in two-point plays. This DataFrame is assigned to the team_season_and1_lowest_committers_2pt variable.

215. `team_season_and1_lowest_committers_2pt`
Outputs the team_season_and1_lowest_committers_2pt DataFrame, which contains the top 50 rows representing the opposing teams and seasons with the lowest number of and-1 situations committed in two-point plays.

216. `team_season_and1_highest_committers_3pt = team_and1_commits_by_season.sort_values(by=['And_1s_3pt' 'Shooting_Fouls'], ascending=[False, True]).head(50)`
Sorts the team_and1_commits_by_season DataFrame by the 'And_1s_3pt' and 'Shooting_Fouls' columns in descending order for 'And_1s_3pt' and ascending order for 'Shooting_Fouls'. The resulting DataFrame contains the top 50 rows, representing the opposing teams and seasons with the highest number of and-1 situations committed in three-point plays. This DataFrame is assigned to the team_season_and1_highest_committers_3pt variable.

217. `team_season_and1_highest_committers_3pt`
Outputs the team_season_and1_highest_committers_3pt DataFrame, which contains the top 50 rows representing the opposing teams and seasons with the highest number of and-1 situations committed in three-point plays.

218. `team_season_and1_lowest_committers_3pt = team_and1_commits_by_season.sort_values(by=['And_1s_3pt', 'Shooting_Fouls'], ascending=[True, True]).head(50)`
Sorts the team_and1_commits_by_season DataFrame by the 'And_1s_3pt' and 'Shooting_Fouls' columns in ascending order for both columns. The resulting DataFrame contains the top 50 rows, representing the opposing teams and seasons with the lowest number of and-1 situations committed in three-point plays. This DataFrame is assigned to the team_season_and1_lowest_committers_3pt variable.

219. `team_season_and1_lowest_committers_3pt`
Outputs the team_season_and1_lowest_committers_3pt DataFrame, which contains the top 50 rows representing the opposing teams and seasons with the lowest number of and-1 situations committed in three-point plays.

220. `team_and1_commits_by_season['And_1+'] = (team_and1_commits_by_season['Season'].apply(lambda x: season_avg_and1s_dict[x]) / team_and1_commits_by_season['And_1s'] * 100).round(2)`
Calculates the and-1 plus percentage for each season in the team_and1_commits_by_season DataFrame. The lambda function is applied to the 'Season' column, retrieving the average number of and-1 situations per game from the season_avg_and1s_dict dictionary. This value is divided by the 'And_1s' column, representing the total number of and-1 situations committed by the team in each season. The resulting percentage is then multiplied by 100 and rounded to 2 decimal places. The calculated values are assigned to the 'And_1+' column in the DataFrame.

221. `team_and1_commits_by_season`
Outputs the team_and1_commits_by_season DataFrame, which now includes the 'And_1+' column representing the and-1 plus percentage for each season.

222. `team_season_and1_highest_committers_total = team_and1_commits_by_season.sort_values(by=['And_1+', 'Shooting_Fouls'], ascending=[True, True]).head(50)`
Retrieves the top 50 seasons with the highest and-1 plus percentage and the lowest number of shooting fouls committed by the team. The DataFrame team_and1_commits_by_season is sorted in ascending order based on the 'And_1+' column (representing the and-1 plus percentage) and the 'Shooting_Fouls' column. The head() function is then used to select the top 50 rows with the highest values. The resulting DataFrame is assigned to the variable team_season_and1_highest_committers_total.

223. `team_season_and1_highest_committers_total`
Outputs the team_season_and1_highest_committers_total DataFrame, which contains the top 50 seasons with the highest and-1 plus percentage and the lowest number of shooting fouls committed by the team.

224. `team_season_and1_lowest_committers_total = team_and1_commits_by_season.sort_values(by=['And_1+', 'Shooting_Fouls'], ascending=[False, True]).head(50)`
Retrieves the top 50 seasons with the lowest and-1 plus percentage and the lowest number of shooting fouls committed by the team. The DataFrame team_and1_commits_by_season is sorted in descending order based on the 'And_1+' column (representing the and-1 plus percentage) and the 'Shooting_Fouls' column. The head() function is then used to select the top 50 rows with the lowest values. The resulting DataFrame is assigned to the variable team_season_and1_lowest_committers_total.

225. `team_season_and1_lowest_committers_total`
Outputs the team_season_and1_lowest_committers_total DataFrame, which contains the top 50 seasons with the lowest and-1 plus percentage and the lowest number of shooting fouls committed by the team.

226. `team_and1_commits_by_season['Team_ID'] = (team_and1_commits_by_season['Opposing_Team'].apply(lambd x: team_ids_dict[x]))`
Assigns the corresponding team ID to each opposing team in the team_and1_commits_by_season DataFrame using the team_ids_dict dictionary. The lambda function is applied to the 'Opposing_Team' column, mapping each team name to its corresponding ID in the dictionary. The resulting IDs are then assigned to the 'Team_ID' column.

227. `team_and1_commits_by_season`
Outputs the updated team_and1_commits_by_season DataFrame, which now includes the 'Team_ID' column with the corresponding team IDs for each opposing team.

228. `file_path = 'team_yearly_stats.csv'`
Specifies the file path of the 'team_yearly_stats.csv' file.

229. `nba_yearly_stats = pd.read_csv(file_path)`
Reads the CSV file located at the specified file path into a pandas DataFrame named 'nba_yearly_stats'. This DataFrame contains the yearly stats of NBA teams.

230. `for index, row in team_and1_commits_by_season.iterrows():`
Iterates over each row in the 'team_and1_commits_by_season' DataFrame.

231. `team = team_ids_dict[row['Opposing_Team']]`
Retrieves the team ID from the 'team_ids_dict' dictionary based on the 'Opposing_Team' value in the current row.

232. `season = row['Season']`
Retrieves the season value from the 'Season' column in the current row.

233. `yearly_row = nba_yearly_stats.loc[(nba_yearly_stats['TEAM_ID'] == team) (nba_yearly_stats['YEAR' == season)]`
Filters the 'nba_yearly_stats' DataFrame based on the conditions that the 'TEAM_ID' column matches the 'team' value and the 'YEAR' column matches the 'season' value. The result is stored in the 'yearly_row' variable.

234. `team_and1_commits_by_season.at[index, 'Win_Pct'] = yearly_row['WIN_PCT'].values[0]`
Assigns the value of the 'WIN_PCT' column from the 'yearly_row' DataFrame to the corresponding row and 'Win_Pct' column in the 'team_and1_commits_by_season' DataFrame.

235. `team_and1_commits_by_season.at[index, 'PF_Per_Game'] = yearly_row['PF'].values[0]`

Assigns the value of the 'PF' column from the 'yearly_row' DataFrame to the corresponding row and 'PF_Per_Game' column in the 'team_and1_commits_by_season' DataFrame.

`and1_plus = team_and1_commits_by_season['And_1+']`
Retrieves the 'And_1+' column from the 'team_and1_commits_by_season' DataFrame and assigns it to the variable 'and1_plus'.

`win_pct = team_and1_commits_by_season['Win_Pct']`
Retrieves the 'Win_Pct' column from the 'team_and1_commits_by_season' DataFrame and assigns it to the variable 'win_pct'.

`plt.scatter(and1_plus, win_pct)`
Creates a scatter plot with 'and1_plus' on the x-axis and 'win_pct' on the y-axis.

`plt.xlabel('And1+')`
Sets the label for the x-axis as 'And1+'.

`plt.ylabel('Win %')`
Sets the label for the y-axis as 'Win %'.

`plt.show()`
Displays the scatter plot.

`slope, intercept, r_value, p_value, std_err = stats.linregress(and1_plus, win_pct)`
Performs linear regression analysis on 'and1_plus' and 'win_pct', and assigns the slope, intercept, correlation coefficient (r-value), p-value, and standard error to the respective variables.

`print("Slope:", slope)`
Prints the slope.

`print("Intercept:", intercept)`
Prints the intercept.

`print("R-squared:", r_value**2)`
Prints the R-squared value.

`print("p-value:", p_value)`
Prints the p-value.

`print("Standard Error:", std_err)`
Prints the standard error.

`and1_plus = team_and1_commits_by_season['And_1+']`
Retrieves the 'And_1+' column from the 'team_and1_commits_by_season' DataFrame and assigns it to the variable 'and1_plus'.

`pf_per_game = team_and1_commits_by_season['PF_Per_Game']`
Retrieves the 'PF_Per_Game' column from the 'team_and1_commits_by_season' DataFrame and assigns it to the variable 'pf_per_game'.

`plt.scatter(and1_plus, pf_per_game)`
Creates a scatter plot with 'and1_plus' on the x-axis and 'pf_per_game' on the y-axis.

`plt.xlabel('And1+')`
Sets the label for the x-axis as 'And1+'.

`plt.ylabel('Personal Fouls per Game')`
Sets the label for the y-axis as 'Personal Fouls per Game'.

`plt.show()`
Displays the scatter plot.

`slope, intercept, r_value, p_value, std_err = stats.linregress(and1_plus, pf_per_game)`
Performs linear regression analysis on 'and1_plus' and 'pf_per_game', and assigns the slope, intercept, correlation coefficient (r-value), p-value, and standard error to the respective variables.

`print("Slope:", slope)`

Prints the slope.

```
258.  print("Intercept:", intercept)
```
Prints the intercept.

```
259.  print("R-squared:", r_value**2)
```
Prints the R-squared value.

```
260.  print("p-value:", p_value)
```
Prints the p-value.

```
261.  print("Standard Error:", std_err)
```
Prints the standard error.

2 Packages and APIs

2.1 Time

From: Python Standard Library

Import Statement: import time

Description: We used the sleep function of this library. This was used for making delays to ensure that the requests to the nba_api did not timeout.

2.2 Re

From: Standard Python Library

Import Statement: import re

Description: This library handles regular expressions. We used this to search for patterns in text such as 3PT field goals.

2.3 DateTime

From: Python Standard Library

Import Statement: from datetime import datetime

Description: This library handles date and time in Python. This was used for elapsed time functions.

2.4 Pandas

From: Pandas Library

Import Statement: import pandas as pd

Description: Pandas lets us use Dataframes, which are the tables you see throughout our program. We can display and manipulate tables using this.

2.5 Requests

From: Requests Library

Import Statement: import requests

Description: This library allows us to make requests from an api url.

2.6 Pyplot

From: Matlab-Style Plotting Library

Import Statement: import matplotlib.pyplot as plt

Description: This library allows us to make and display plots and other forms of visual data.

2.7 Stats

From: Scientific Python Library

Import Statement: import scipy.stats as stats

Description: This library allows us to use various statistics functions such as mean.

2.8 Random

From: Scientific Python Library

Import Statement: import random

Description: This library allows us to use a random number generator. We use this to choose a random time near 0.7 seconds to sleep to avoid timeouts.

2.9 Os

From: Scientific Python Library

Import Statement: import os

Description: This library allows us to use operating system tools. We used it to save files to the computer.

2.10 Teams

From: NBA API Static Stats Library

Import Statement: from nba_api.stats.static import teams

Description: This class from the nba_api gives us team data such as a team's name and unique ID. [2]

2.11 Players

From: NBA API Static Stats Library

Import Statement: from nba_api.stats.static import players

Description: This class from the nba_api gives us player data such as ID and name. [2]

2.12 TeamGameLogs

From: NBA API Stats Endpoint Library

Import Statement: from nba_api.stats.endpoints import TeamGameLogs

Description: This class from the nba_api gives us data from each game by team. [6]

2.13 TeamYearByYearStats

From: NBA API Stats Endpoint Library

Import Statement: from nba_api.stats.endpoints import TeamYearByYearStats

Description: This class from the nba_api gives us stats for each team for a given year or years. [6]

2.14 PlayByPlayV2

From: NBA API Stats Endpoint Library

Import Statement: from nba_api.stats.endpoints import PlayByPlayV2 as pbp2

Description: This class gives us play by play data for any game we specify. We used this to calculate and-1 opportunities. [6]

2.15 PlayerDashboardByYearOverYear

From: NBA API Stats Endpoint Library

Import Statement: from nba_api.stats.endpoints import PlayerDashboardByYearOverYear

Description: This class is used to access a player's performance statistics broken down year by year.

2.16 PlayerCareerStats

From: NBA API Stats Endpoint Library

Import Statement: from nba_api.stats.endpoints import PlayerCareerStats

Description: This class is used to access a player's career stats. It provides a summary of the player's entire career in the NBA, including total points, rebounds, assists, and more.

2.17 Leaguegamefinder

From: NBA API Stats Endpoint Library

Import Statement: from nba_api.stats.endpoints import leaguegamefinder

Description: This class is used to find NBA games based on various criteria. You can use game_id to find games, or find games with a certain amount of point scored, etc. [6]

3 Findings

The main purpose of this project was to analyze and-1 plays throughout the history of the NBA. We define an "and-1 play" as any play where an offensive player takes a shot, gets fouled in the process, and still makes the shot. The player then receives one free throw in addition to the points they have already scored due to the made basket. When we refer to "and-1 plays" throughout this report, we are not concerned with whether or not the player made the and-1 free throw following the basket. We are just referring to any time a player made the shot after being fouled. In order to analyze these and-1 plays, we needed to access large amounts of data from the NBA API [5] using Python and GitHub [1]. There, we were able to find every play that has occurred in the NBA since the 1996-1997 season [5]. In order to find every instance of an and-1 play in those years, we first had to download all of the data into a CSV that contained every single play from every single one of those games. This took many hours of API requests and when it was all collected, the file containing all the details from all these plays ended up being around 3GB of CSV data. Once we had all that data, we could then filter it into just the instances of and-1 plays. So from the large file of play by play data, we created a new CSV dataset of all the plays where an and-1 play occurred. This included the offensive player taking the shot and making it, the defensive player on the other team committing a shooting foul, and the offensive player taking the free throw. Once we had this much smaller, much more specific dataset, we were able to analyze all of the and-1 plays in the NBA play by play tracking era. The following analysis comes from that dataset.

3.1 General

One of the main goals of this project was to find out how often and-1 plays actually occur. We determined exactly what percent of the time a player makes the basket when they are fouled. In other words, how often a shooting foul results in an and-1 play. We calculated this for shots of all types, 2pt shots, and 3pt shots and the results can be found in Figure 1. And-1 plays do not happen extremely often as we discovered, seeing that on any type of shot, the player makes the shot after being fouled less that 23% of the time.

```
Shooting Fouls That Result in and And-1 Play:          22.04% (137221 of 622563)
2pt Shooting Fouls That Result in and And-1 Play:      22.07% (134154 of 607740)
3pt Shooting Fouls That Result in and And-1 Play:      20.69% (3067 of 14823)
```

Figure 1: Percentage of total shots, 2pt shots, and 3pt shots that result in and-1 plays.

3.2 Individual Performance

Another aspect of and-1 plays that we wanted to analyze was the success that individual players have had when it comes to these plays both all time, and in any given season.

3.2.1 All-Time And-1 Leaders

We looked at the players with the most and-1 plays since the 1996-1997 NBA season on all shot types (Figure 2), 2pt shots (Figure 3), and 3pt shots (Figure 4). We found that many of the total and 2pt and-1 leaders remain the same, whereas the 3pt and-1 leaders are very different. LeBron James has the most total and 2pt and-1 plays of all time, with Shaquille O'Neal, Dwight Howard, Kobe Bryant, and Tim Duncan rounding out the top 5. James Harden has the most 3pt and-1 plays of all time with Jamal Crawford, JJ Redick, Stephen Curry, and Damian Lillard rounding out the top 5.

Player Name	And_1s	Shooting_Fouls	And_1s_2pt	And_1s_3pt
LeBron James	1406	4842	1398	8
Shaquille O'Neal	1075	3718	1075	0
Dwight Howard	888	3975	888	0
Kobe Bryant	836	3820	819	17
Tim Duncan	834	3884	834	0
Dwyane Wade	723	3276	718	5
Paul Pierce	711	3382	699	12
Amar'e Stoudemire	693	2707	693	0
DeMar DeRozan	676	3007	667	9
Giannis Antetokounmpo	664	2622	664	0
Dirk Nowitzki	660	3067	632	28
Russell Westbrook	646	3089	638	8
Carmelo Anthony	641	3103	628	13
Kevin Durant	627	2864	598	29
James Harden	621	3374	521	100
Allen Iverson	618	3126	604	14
Elton Brand	602	2236	602	0
Pau Gasol	592	2569	590	2
Zach Randolph	509	2002	509	0
Chris Bosh	504	2582	503	1

Figure 2: Top 20 all-time and-1 leaders on all shot types in NBA history.

Player Name	And_1s_2pt	Shooting_Fouls	And_1s	And_1s_3pt
LeBron James	1398	4842	1406	8
Shaquille O'Neal	1075	3718	1075	0
Dwight Howard	888	3975	888	0
Tim Duncan	834	3884	834	0
Kobe Bryant	819	3820	836	17
Dwyane Wade	718	3276	723	5
Paul Pierce	699	3382	711	12
Amar'e Stoudemire	693	2707	693	0
DeMar DeRozan	667	3007	676	9
Giannis Antetokounmpo	664	2622	664	0
Russell Westbrook	638	3089	646	8
Dirk Nowitzki	632	3067	660	28
Carmelo Anthony	628	3103	641	13
Allen Iverson	604	3126	618	14
Elton Brand	602	2236	602	0
Kevin Durant	598	2864	627	29
Pau Gasol	590	2569	592	2
James Harden	521	3374	621	100
Zach Randolph	509	2002	509	0
Chris Bosh	503	2582	504	1

Figure 3: Top 20 all-time and-1 leaders on 2pt shots in NBA history.

Player Name	And_1s_3pt	Shooting_Fouls	And_1s	And_1s_2pt
James Harden	100	3374	621	521
Jamal Crawford	62	1521	347	285
JJ Redick	59	701	142	83
Stephen Curry	56	1313	347	291
Damian Lillard	38	1722	360	322
Jordan Clarkson	34	689	194	160
Terrence Ross	33	379	106	73
Kyle Lowry	33	1596	336	303
Tim Hardaway Jr.	29	655	164	135
Nick Young	29	674	141	112
Kevin Durant	29	2864	627	598
Khris Middleton	28	793	175	147
Dirk Nowitzki	28	3067	660	632
Kyle Korver	26	257	72	46
Reggie Miller	25	767	148	123
Kemba Walker	25	1266	284	259
Davis Bertans	24	133	38	14
Jae Crowder	24	557	123	99
Nicolas Batum	24	681	171	147
Deron Williams	23	1302	300	277

Figure 4: Top 20 all-time and-1 leaders on 3pt shots in NBA history.

3.2.2 Single-Season And-1 Leaders

Now that we have seen which players have had the most and-1 plays of all time. We can now look at which players had the most and-1 plays in a single season. We examined the most and-1 plays in a single season on all shot types (Figure 5), 2pt shots (Figure 6), and 3pt shots (Figure **??**). Once again the total and 2pt leaders were very similar, whereas the 3pt leaders were very different. Shaquille O'Neal has 3 of the top 5 and-1 seasons of all time (2000-2001, 1999-2000, and 2001-2002) with Amare Stoudemire's 2007-2008 season and LeBron James' 2005-2006 season making up the rest of the top 5. James Harden owns the top 4 3pt and-1 seasons of all time (2018-2019, 2017-2018, 2019-2020, 2021-2022) with Jamal Crawford's 2008-2009 season barely sneaking into the top 5 with the 5th spot.

Player Name	Season	And_1s	Shooting_Fouls	And_1s_2pt	And_1s_3pt
Shaquille O'Neal	2000-01	114	401	114	0
Shaquille O'Neal	1999-00	110	346	110	0
Amar'e Stoudemire	2007-08	109	366	109	0
Shaquille O'Neal	2001-02	107	330	107	0
LeBron James	2005-06	106	357	106	0
Giannis Antetokounmpo	2018-19	99	342	99	0
LeBron James	2008-09	97	369	96	1
Amar'e Stoudemire	2006-07	96	311	96	0
LeBron James	2006-07	96	316	96	0
Giannis Antetokounmpo	2022-23	96	379	96	0
Shaquille O'Neal	2002-03	94	339	94	0
Amar'e Stoudemire	2004-05	94	401	94	0
Giannis Antetokounmpo	2019-20	93	302	93	0
Dwight Howard	2008-09	93	373	93	0
Shaquille O'Neal	1997-98	92	312	92	0
Shaquille O'Neal	2004-05	89	325	89	0
Karl Malone	1996-97	88	306	88	0
LeBron James	2007-08	88	342	87	1
Shaquille O'Neal	2003-04	86	292	86	0
Giannis Antetokounmpo	2021-22	84	364	84	0

Figure 5: Top 20 single-season and-1 leaders on all shot types in NBA history.

Player Name	Season	And_1s_2pt	Shooting_Fouls	And_1s	And_1s_3pt
Shaquille O'Neal	2000-01	114	401	114	0
Shaquille O'Neal	1999-00	110	346	110	0
Amar'e Stoudemire	2007-08	109	366	109	0
Shaquille O'Neal	2001-02	107	330	107	0
LeBron James	2005-06	106	357	106	0
Giannis Antetokounmpo	2018-19	99	342	99	0
Amar'e Stoudemire	2006-07	96	311	96	0
LeBron James	2006-07	96	316	96	0
LeBron James	2008-09	96	369	97	1
Giannis Antetokounmpo	2022-23	96	379	96	0
Shaquille O'Neal	2002-03	94	339	94	0
Amar'e Stoudemire	2004-05	94	401	94	0
Giannis Antetokounmpo	2019-20	93	302	93	0
Dwight Howard	2008-09	93	373	93	0
Shaquille O'Neal	1997-98	92	312	92	0
Shaquille O'Neal	2004-05	89	325	89	0
Karl Malone	1996-97	88	306	88	0
LeBron James	2007-08	87	342	88	1
Shaquille O'Neal	2003-04	86	292	86	0
Giannis Antetokounmpo	2021-22	84	364	84	0

Figure 6: Top 20 single-season and-1 leaders on 2pt shots in NBA history.

Player Name	Season	And_1s_3pt	Shooting_Fouls	And_1s	And_1s_2pt
James Harden	2018-19	20	356	76	56
James Harden	2017-18	14	271	55	41
James Harden	2019-20	14	321	74	60
James Harden	2021-22	13	232	52	39
Jamal Crawford	2008-09	12	117	35	23
Garrison Mathews	2021-22	11	60	14	3
Terrence Ross	2019-20	11	68	22	11
Stephen Curry	2022-23	11	111	37	26
Terrence Ross	2018-19	10	50	15	5
Darius Miller	2018-19	9	28	10	1
JJ Redick	2018-19	9	71	14	5
Jamal Crawford	2009-10	9	116	26	17
Kyle Lowry	2016-17	9	146	39	30
James Harden	2022-23	9	150	36	27
Davis Bertans	2020-21	8	29	9	1
Landry Shamet	2018-19	8	33	13	5
Jordan Clarkson	2019-20	8	70	25	17
Jae Crowder	2018-19	8	77	18	10
Stephen Curry	2018-19	8	108	31	23
Nick Young	2013-14	8	120	34	26

Figure 7: Top 20 single-season and-1 leaders on 3pt shots in NBA history.

3.2.3 And-1 Percentages

We have already looked at the all-time and single season leaders in every type of and-1 play, which lets us know who drew the most and-1's. But now we want to look at which players were the most effective and least effective at making shots after they had been fouled. We called this and-1 percentage and it tells what percent of the time a player receives a shooting foul, they make the shot and receive and and-1. We calculated the percentage of shooting fouls of any type that resulted in and-1's for players with 100 or more and-1 plays in their career using the formula $and1\ pct = \frac{and1s}{all\ shooting\ fouls}$. We looked at both the best (Figure 8) and worst (Figure 9) of these percentages. We then calculated the percentage of 2pt shooting fouls that resulted in and-1's for players with 100 or more and-1 plays in their career using the formula $2pt\ and1\ pct = \frac{2pt\ and1s}{2pt\ shooting\ fouls}$. We looked at both the best (Figure 10) and worst (Figure 11) of these percentages. Finally we calculated the percentage of 3pt shooting fouls that resulted in and-1's for players with 3 or more 3pt and-1 plays in their career using the formula $3pt\ and1\ pct = \frac{3pt\ and1s}{3pt\ shooting\ fouls}$. We calculated both the best (Figure 12) and worst (Figure 13) of these percentages.

Once again we found that the overall and 2pt and-1 percentages were similar. The highest and-1 percentages belonged to Montrezl Harrel, Hassan Whiteside, and Jakob Poeltl, while the worst belonged to Elden Campbell, Theo Ratliff, and Danilo Gallinari. It is suspected the higher percentages belong to newer players or players with shorter careers because they were not fouled as often and therefore could have just made a lot of shots in their small sample size of fouls. They are also all big men which means most of their shots are closer to the basket, so they are more likely to make shots they are fouled on due to proximity. We wanted to limit the search space for 3pt and-1 percentage because we know that there are many players who have been fouled very few times on 3 pointers and made a large percent of them. For example, there were many players who had only been fouled 1 or 2 times on 3pt shots and made either all or half of them, so their 3pt and-1 percentages were very high. After filtering, it is found that Frank Nkilitina was the only player to make all of his 3pt shots where he was fouled (min. 3 attempts). The players shooting 75% or better are Quentin Grimes, Moses Moody, and Terry Mills. The players with the worst 3pt and-1 percentages were Mo Williams, Jameer Nelson,

Lou Williams, Jarret Jack, and Kevin Love. These players had many more 3pt shots that they were fouled on than the leaders, so perhaps a higher minimum number of 3pt shooting fouls would benefit the search for effectiveness.

Player Name	And_1s	Shooting_Fouls	And_1_Percentage
Montrezl Harrell	319	913	0.349398
Hassan Whiteside	262	813	0.322263
Jakob Poeltl	122	383	0.318538
T.J. Warren	129	406	0.317734
Nikola Jokic	329	1040	0.316346
Richaun Holmes	118	388	0.304124
Mikal Bridges	119	399	0.298246
Al Jefferson	434	1460	0.297260
John Collins	158	537	0.294227
Keldon Johnson	121	414	0.292271
Elfrid Payton	142	486	0.292181
Karl-Anthony Towns	345	1184	0.291385
Boris Diaw	174	599	0.290484
LeBron James	1406	4842	0.290376
Shaquille O'Neal	1075	3718	0.289134
JaVale McGee	218	754	0.289125
Jalen Brunson	111	386	0.287565
Mike Miller	189	660	0.286364
Terry Rozier	120	420	0.285714
Wilson Chandler	175	614	0.285016

Figure 8: Top 20 and-1 percentage leaders on all shot types in NBA history (min. 100 and-1 plays).

Player Name	And_1s	Shooting_Fouls	And_1_Percentage
Elden Campbell	132	1099	0.120109
Theo Ratliff	120	799	0.150188
Danilo Gallinari	190	1259	0.150913
Reggie Evans	120	787	0.152478
Bob Sura	104	674	0.154303
Andrei Kirilenko	238	1539	0.154646
Ricky Rubio	116	748	0.155080
Chauncey Billups	257	1642	0.156516
Malik Rose	118	753	0.156707
Clar. Weatherspoon	110	701	0.156919
Keith Van Horn	141	884	0.159502
David Wesley	122	756	0.161376
Sam Cassell	181	1107	0.163505
Antonio Davis	194	1185	0.163713
Jerry Stackhouse	290	1769	0.163934
Kevin Love	262	1595	0.164263
Joakim Noah	139	840	0.165476
Corey Maggette	379	2267	0.167181
Zydrunas Ilgauskas	251	1490	0.168456
Chris Andersen	111	656	0.169207

Figure 9: 20 lowest and-1 percentages on all shot types in NBA history (min. 100 and-1 plays).

Player Name	And_1s_2pt	Shooting_Fouls_2pt	And_1_Percentage_2pt
Montrezl Harrell	319	913	0.349398
Hassan Whiteside	262	813	0.322263
Nikola Jokic	326	1021	0.319295
Jakob Poeltl	122	383	0.318538
T.J. Warren	126	396	0.318182
Richaun Holmes	118	388	0.304124
Mikal Bridges	116	384	0.302083
Al Jefferson	434	1460	0.297260
John Collins	158	533	0.296435
Jalen Brunson	109	368	0.296196
Keldon Johnson	119	404	0.294554
Elfrid Payton	142	484	0.293388
Karl-Anthony Towns	339	1160	0.292241
Terrence Ross	73	250	0.292000
LeBron James	1398	4792	0.291736
Boris Diaw	174	597	0.291457
Shaquille O'Neal	1075	3718	0.289134
JaVale McGee	218	754	0.289125
Terry Rozier	105	364	0.288462
Mike Miller	187	650	0.287692

Figure 10: Top 20 and-1 percentage leaders on 2pt shots in NBA history (min. 100 and-1 plays).

Player Name	And_1s_2pt	Shooting_Fouls_2pt	And_1_Percentage_2pt
Elden Campbell	132	1099	0.120109
Danilo Gallinari	181	1209	0.149711
Theo Ratliff	120	799	0.150188
Andrei Kirilenko	231	1522	0.151774
Reggie Evans	120	787	0.152478
Bob Sura	102	661	0.154312
Malik Rose	118	753	0.156707
Clar. Weatherspoon	110	701	0.156919
Chauncey Billups	245	1556	0.157455
Ricky Rubio	114	717	0.158996
JJ Redick	83	518	0.160232
Keith Van Horn	141	877	0.160775
Klay Thompson	99	608	0.162829
Sam Cassell	180	1104	0.163043
Antonio Davis	194	1185	0.163713
Jerry Stackhouse	285	1739	0.163887
David Wesley	121	738	0.163957
Joakim Noah	139	840	0.165476
Zydrunas Ilgauskas	251	1489	0.168570
Corey Maggette	378	2242	0.168599

Figure 11: 20 lowest and-1 percentages on 2pt shots in NBA history (min. 100 and-1 plays).

Player Name	And_1s_3pt	Shooting_Fouls_3pt	And_1_Percentage_3pt
Frank Ntilikina	3	3	1.000000
Quentin Grimes	5	6	0.833333
Moses Moody	3	4	0.750000
Terry Mills	3	4	0.750000
Donald Sloan	4	6	0.666667
Andrea Bargnani	5	8	0.625000
Malachi Flynn	3	5	0.600000
Lance Thomas	3	5	0.600000
Jalen Rose	4	8	0.500000
Steve Francis	4	8	0.500000
Seth Curry	20	42	0.476190
Dejounte Murray	7	15	0.466667
Darius Miller	21	46	0.456522
Raja Bell	5	11	0.454545
Keyon Dooling	5	11	0.454545
Wilson Chandler	10	22	0.454545
Eddie House	4	9	0.444444
Austin Reaves	4	9	0.444444
Joe Dumars	3	7	0.428571
MarShon Brooks	3	7	0.428571

Figure 12: Top 20 and-1 percentage leaders on 3pt shots in NBA history (min. 3 and-1 3pt plays).

Player Name	And_1s_3pt	Shooting_Fouls_3pt	And_1_Percentage_3pt
Mo Williams	3	40	0.075000
Jameer Nelson	3	39	0.076923
Lou Williams	19	243	0.078189
Jarrett Jack	3	34	0.088235
Kevin Love	9	101	0.089109
Kevin Martin	5	52	0.096154
Kevin Porter Jr.	3	31	0.096774
Russell Westbrook	8	82	0.097561
Wesley Matthews	5	51	0.098039
Zach LaVine	3	30	0.100000
Tracy McGrady	11	106	0.103774
D.J. Augustin	10	95	0.105263
Mike Conley	10	82	0.121951
Kobe Bryant	17	135	0.125926
Trae Young	13	103	0.126214
Jeff Teague	5	39	0.128205
Jason Kidd	4	31	0.129032
Corey Brewer	3	23	0.130435
Marcus Smart	8	61	0.131148
Ben McLemore	5	38	0.131579

Figure 13: 20 lowest and-1 percentages on 3pt shots in NBA history (min. 3 and-1 3pt plays).

3.3 Offensive Team Performance

Once we were finished with players effectiveness at drawing and-1 plays, we wanted to look at how well individual teams drew and-1 plays. We did this for both all time teams, and teams in a single season.

3.3.1 All-Time

While this information may not mean all that much when it comes to analyzing and-1's, we thought it would be interesting to see which NBA franchises have drawn the most and-1 plays since 1996. We once again looked at teams that have drawn the most and-1 plays on all shot types (Figure 14), 2pt shots (Figure 15), and 3pt shots (Figure 16). What we found is that the Los Angeles Lakers have drawn the most and-1 plays on both all shot types and 2pt shots. The Washington Wizards, Denver Nuggets, Sacramento Kings, and Los Angeles Clippers make up the rest of the top 5 in drawing and-1 plays on both all shot types and just 2pt shots. When it comes to drawing and-1 plays on 3pt shots, the Houston Rockets are the best team of all time, with the Golden State Warriors, Philadelphia 76ers, Los Angeles Lakers, and Portland Trail Blazers following in the top 5.

Team	And_1s	Shooting_Fouls	And_1s_2pt	And_1s_3pt
LAL	5383	22615	5258	125
WAS	5000	21808	4891	109
DEN	4910	21897	4796	114
SAC	4825	21448	4749	76
LAC	4740	21297	4624	116
UTA	4706	22036	4612	94
ORL	4677	20896	4562	115
PHX	4675	20087	4590	85
NYK	4613	20620	4515	98
CLE	4589	21149	4512	77
HOU	4580	21407	4416	164
SAS	4561	20410	4449	112
MIN	4547	21049	4482	65
MIA	4536	20249	4450	86
PHI	4532	21997	4397	135
GSW	4525	20455	4381	144
IND	4525	20544	4407	118
CHI	4485	20417	4400	85
TOR	4483	20730	4375	108
MIL	4473	20367	4371	102

Figure 14: Top 20 teams with the most and-1s on all shot types in NBA history.

Team	And_1s_2pt	Shooting_Fouls	And_1s	And_1s_3pt
LAL	5258	22615	5383	125
WAS	4891	21808	5000	109
DEN	4796	21897	4910	114
SAC	4749	21448	4825	76
LAC	4624	21297	4740	116
UTA	4612	22036	4706	94
PHX	4590	20087	4675	85
ORL	4562	20896	4677	115
NYK	4515	20620	4613	98
CLE	4512	21149	4589	77
MIN	4482	21049	4547	65
MIA	4450	20249	4536	86
SAS	4449	20410	4561	112
HOU	4416	21407	4580	164
IND	4407	20544	4525	118
CHI	4400	20417	4485	85
PHI	4397	21997	4532	135
GSW	4381	20455	4525	144
TOR	4375	20730	4483	108
MIL	4371	20367	4473	102

Figure 15: Top 20 teams with the most and-1s on 2pt shots in NBA history.

Team	And_1s_3pt	Shooting_Fouls	And_1s_2pt	And_1s
HOU	164	21407	4416	4580
GSW	144	20455	4381	4525
PHI	135	21997	4397	4532
LAL	125	22615	5258	5383
POR	121	20129	4285	4406
IND	118	20544	4407	4525
LAC	116	21297	4624	4740
ORL	115	20896	4562	4677
DEN	114	21897	4796	4910
SAS	112	20410	4449	4561
WAS	109	21808	4891	5000
TOR	108	20730	4375	4483
DAL	107	19483	4195	4302
BKN	102	8566	1936	2038
MIL	102	20367	4371	4473
DET	99	20779	4326	4425
NYK	98	20620	4515	4613
UTA	94	22036	4612	4706
ATL	93	20404	4224	4317
CHA	92	15362	3397	3489

Figure 16: Top 20 teams with the most and-1s on 3pt shots in NBA history.

3.3.2 Single-Season

Getting a little more specific than all-time rankings, we wanted to see which individual teams drew the most and-1 plays in a single season. We calculated the best single seasons for drawing and-1's on all shot types (Figure 17), the worst on all shot types (Figure 18), the best on 2pt shots (Figure 19), the worst on 2pt shots (Figure 20), the best on 3pt shots (Figure 21), and the worst on 3pt shots (Figure 22).

We found that the 1997-1998 Lakers drew more and-1 plays than any other team in a single season in history. The 2022-2023 Knicks, 2022-2023 Lakers, 2022-2023 Hornets, and 2022-2023 Timberwolves finished out the top 5. The 1998-1999 Bulls drew the least and-1 plays out of any team all time, followed by the 1998-1999 Warriors, 1998-1999 Hornets, 1998-1999 Raptors, and 1998-1999 Timberwolves. When it comes to 3pt shots, the best team at drawing and-1 plays was the 2018-2019 Rockets, followed by the 2017-2018 Rockets, 2018-2019 76ers, 2022-2023 Warriors, and 2021-2022 Nets. There is no point in analyzing who had the lease because over 50 teams have had 0 3pt and-1 plays in a single season.

Team	Season	And_1s	Shooting_Fouls	And_1s_2pt	And_1s_3pt
LAL	1997-98	261	1103	258	3
NYK	2022-23	259	967	252	7
LAL	2022-23	259	971	252	7
CHA	2022-23	251	911	246	5
MIN	2022-23	249	903	245	4
DEN	2010-11	248	1013	246	2
LAC	2018-19	246	942	238	8
CLE	2006-07	244	936	241	3
NYK	2005-06	244	1103	242	2
PHX	2008-09	243	957	243	0
NYK	2006-07	241	1044	237	4
CLE	2005-06	240	992	240	0
MEM	2022-23	239	869	237	2
WAS	2006-07	238	1008	230	8
CHA	2017-18	237	955	224	13
MEM	2009-10	236	940	236	0
SAC	2021-22	235	849	233	2
PHI	2022-23	235	950	223	12
WAS	2022-23	233	831	228	5
SAS	2022-23	231	820	224	7

Figure 17: Top 20 single-season teams with the most and-1s on all shot types in NBA history.

Team	Season	And_1s	Shooting_Fouls	And_1s_2pt	And_1s_3pt
CHI	1998-99	76	479	76	0
GSW	1998-99	79	450	78	1
CHH	1998-99	81	535	81	0
TOR	1998-99	84	520	83	1
MIN	1998-99	85	471	85	0
HOU	1998-99	86	450	86	0
ATL	1998-99	86	503	85	1
PHI	1998-99	86	566	82	4
MIL	1998-99	88	467	88	0
CLE	1998-99	88	480	88	0
NJN	1998-99	88	502	88	0
HOU	2011-12	89	494	87	2
BOS	1998-99	90	427	89	1
MIA	1998-99	92	481	92	0
BOS	2011-12	95	501	93	2
SEA	1998-99	97	495	97	0
DAL	2011-12	97	505	95	2
PHX	1998-99	99	461	99	0
IND	1998-99	101	458	95	6
ORL	1998-99	101	494	99	2

Figure 18: 20 single-season teams with the least and-1s on all shot types in NBA history.

Team	Season	And_1s_2pt	Shooting_Fouls	And_1s	And_1s_3pt
LAL	1997-98	258	1103	261	3
NYK	2022-23	252	967	259	7
LAL	2022-23	252	971	259	7
CHA	2022-23	246	911	251	5
DEN	2010-11	246	1013	248	2
MIN	2022-23	245	903	249	4
PHX	2008-09	243	957	243	0
NYK	2005-06	242	1103	244	2
CLE	2006-07	241	936	244	3
CLE	2005-06	240	992	240	0
LAC	2018-19	238	942	246	8
MEM	2022-23	237	869	239	2
NYK	2006-07	237	1044	241	4
MEM	2009-10	236	940	236	0
SAC	2021-22	233	849	235	2
WAS	2006-07	230	1008	238	8
MIA	2004-05	229	970	231	2
WAS	2022-23	228	831	233	5
LAL	2006-07	227	905	229	2
UTA	2009-10	226	1005	231	5

Figure 19: Top 20 single-season teams with the most and-1s on 2pt shots in NBA history.

Team	Season	And_1s_2pt	Shooting_Fouls	And_1s	And_1s_3pt
CHI	1998-99	76	479	76	0
GSW	1998-99	78	450	79	1
CHH	1998-99	81	535	81	0
PHI	1998-99	82	566	86	4
TOR	1998-99	83	520	84	1
MIN	1998-99	85	471	85	0
ATL	1998-99	85	503	86	1
HOU	1998-99	86	450	86	0
HOU	2011-12	87	494	89	2
MIL	1998-99	88	467	88	0
CLE	1998-99	88	480	88	0
NJN	1998-99	88	502	88	0
BOS	1998-99	89	427	90	1
MIA	1998-99	92	481	92	0
BOS	2011-12	93	501	95	2
IND	1998-99	95	458	101	6
DAL	2011-12	95	505	97	2
SEA	1998-99	97	495	97	0
PHX	1998-99	99	461	99	0
ORL	1998-99	99	494	101	2

Figure 20: 20 single-season teams with the least and-1s on 2pt shots in NBA history.

Team	Season	And_1s_3pt	Shooting_Fouls	And_1s	And_1s_2pt
HOU	2018-19	28	816	192	164
HOU	2017-18	25	813	202	177
PHI	2018-19	22	875	195	173
GSW	2022-23	20	725	194	174
BKN	2021-22	18	763	212	194
ORL	2018-19	16	679	176	160
NOP	2019-20	16	726	185	169
DAL	2019-20	16	754	214	198
BKN	2017-18	16	791	218	202
BKN	2022-23	16	794	224	208
WAS	2020-21	16	830	224	208
ORL	2019-20	15	711	169	154
GSW	2018-19	15	726	183	168
SAS	2019-20	15	738	178	163
TOR	2017-18	15	760	214	199
LAL	2016-17	15	796	193	178
HOU	2019-20	15	800	193	178
UTA	2018-19	15	816	177	162
LAC	2015-16	14	692	147	133
SAS	2018-19	14	750	200	186

Figure 21: Top 20 single-season teams with the most and-1s on 3pt shots in NBA history.

Team	Season	And_1s_3pt	Shooting_Fouls	And_1s	And_1s_2pt
HOU	1998-99	0	450	86	86
PHX	1998-99	0	461	99	99
MIL	1998-99	0	467	88	88
MIN	1998-99	0	471	85	85
CHI	1998-99	0	479	76	76
CLE	1998-99	0	480	88	88
MIA	1998-99	0	481	92	92
DAL	1998-99	0	493	104	104
SEA	1998-99	0	495	97	97
WAS	1998-99	0	497	116	116
NJN	1998-99	0	502	88	88
LAC	1998-99	0	509	121	121
UTA	1998-99	0	511	108	108
CHH	1998-99	0	535	81	81
ATL	2011-12	0	564	134	134
VAN	1998-99	0	566	115	115
NOH	2011-12	0	570	139	139
LAL	1998-99	0	572	152	152
SAS	2011-12	0	580	135	135
NYK	2002-03	0	595	136	136

Figure 22: 20 single-season teams with the least and-1s on 3pt shots in NBA history.

As you can see, all 5 of the worst and-1 drawing seasons occurred in 1998-1999. If we were to go even further, the 9 lowest, and 17 of the 20 lowest and-1 drawing seasons came from teams in 1998-1999. This is due to a lockout that occurred that year, shortening the season and limiting the number of and-1 play possibilities. Similar, the 2022-2023 season had 4 of the top 5 and-1 drawing teams in history. We realized that you could not decide who the best and-1 drawing teams are by comparing them across different seasons, so we had to come up with a way to definitively determine who the best teams were.

Our solution was to create our own stat that measures how good teams were at drawing and-1 plays. We called this stat "And1+", similar to the baseball stat "ERA+" that is used to compare pitcher's ERA's across different seasons. Our And1+ stat compares a teams number of and-1 plays drawn to the league average number of and-1 plays drawn by a single team that single season. This way, we can say "how good was this team at drawing and-1's compared to the rest of the teams in the league that year?". And1+ works like this: an And1+ of 100 is exactly average, meaning you drew the same number of and-1 plays as the average team in the league that year. An And1+ of 105 means that you were 5% better at drawing and-1 plays than the average team that year, and an And1+ of 95 means that you were 5% worse at drawing and-1 plays than the average team that year. So we calculated who had the best And1+ (Figure 23), and the worst And1+ (Figure 24) of all time using this formula: $and1+ = (\frac{and1\ attempts\ that\ season}{league\ avg\ and1\ attempts\ that\ season}) * 100$.

We found that the official greatest team ever at drawing and-1's was the 1998-1999 Lakers with a monumental And1+ of 153.54 meaning they were 53.54% better at drawing and-1's than the average team in the league. They truly had an and-1 play drawing dynasty as 4 of the top 5 and-1 drawing seasons of all time came from the Lakers in 4 of their 5 seasons between 1997 and 2002. The only outlier was the 2020-2021 Wizards with the 3rd best And1+ of all time at 137.70. The official worst team ever at drawing and-1 plays was the 2013-2014 Bucks coming in with an And1+ of 68.69 meaning they were 31.31% worse at drawing and-1 plays than the average team that year. The teams fighting for that top spot as worst of all time are the 2011-2012 Rockets, 2007-2008 Spurs, 2012-2013 76ers, and 2006-2007 Pistons.

Team	Season	And_1+	And_1s	Shooting_Fouls
LAL	1998-99	153.54	152	572
LAL	1997-98	148.09	261	1103
WAS	2020-21	137.70	224	830
LAL	2001-02	135.29	218	834
LAL	1999-00	135.18	223	859
DEN	2010-11	134.66	248	1013
LAL	2000-01	133.49	210	872
GSW	2001-02	132.80	214	921
CHA	2017-18	131.62	237	955
DEN	2012-13	131.11	203	849
UTA	1996-97	130.64	226	894
MIA	2004-05	130.56	231	970
PHX	2008-09	130.27	243	957
NYK	2005-06	130.13	244	1103
NOP	2020-21	129.71	211	850
LAC	2003-04	129.09	202	973
DAL	2019-20	128.66	214	754
LAL	2002-03	128.64	206	879
LAC	2018-19	128.17	246	942
CLE	2005-06	128.00	240	992

Figure 23: Top 20 single-season teams with the highest and1+ in NBA history.

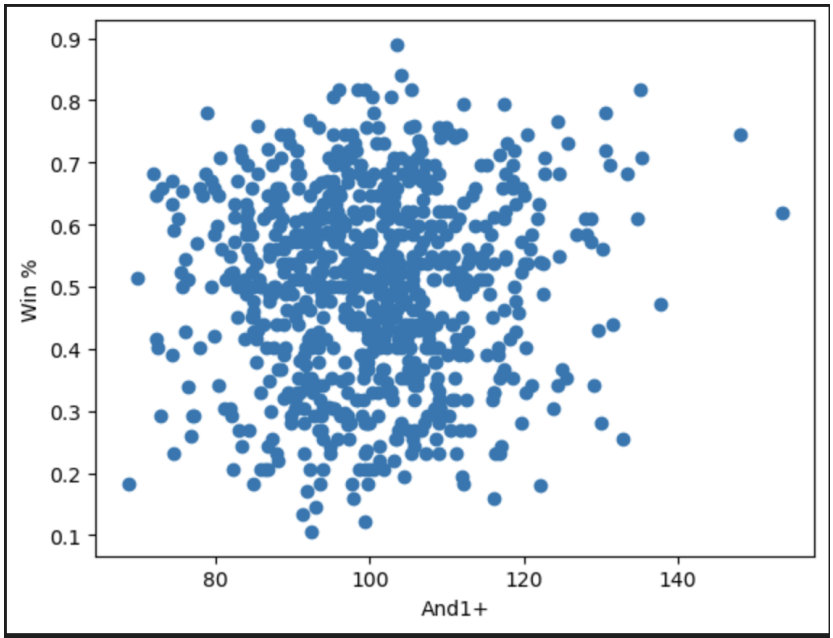
Team	Season	And_1+	And_1s	Shooting_Fouls
MIL	2013-14	68.69	116	663
HOU	2011-12	69.77	89	494
SAS	2007-08	71.82	132	709
PHI	2012-13	72.34	112	592
DET	2006-07	72.36	145	775
DAL	2016-17	72.44	128	572
DAL	2017-18	72.75	131	593
PHX	2005-06	73.07	137	573
DAL	2009-10	74.29	143	748
MIN	2006-07	74.35	149	772
HOU	2006-07	74.35	149	776
BOS	2011-12	74.47	95	501
CLE	2018-19	74.51	143	693
MIN	1999-00	75.17	124	654
ATL	2021-22	75.53	142	694
LAC	2020-21	75.61	123	578
TOR	2007-08	75.63	139	713
MIL	2010-11	76.02	140	757
DAL	2011-12	76.04	97	505
NJN	2004-05	76.30	135	789

Figure 24: Top 20 single-season teams with the lowest and1+ in NBA history.

3.3.3 Correlation to Team Success

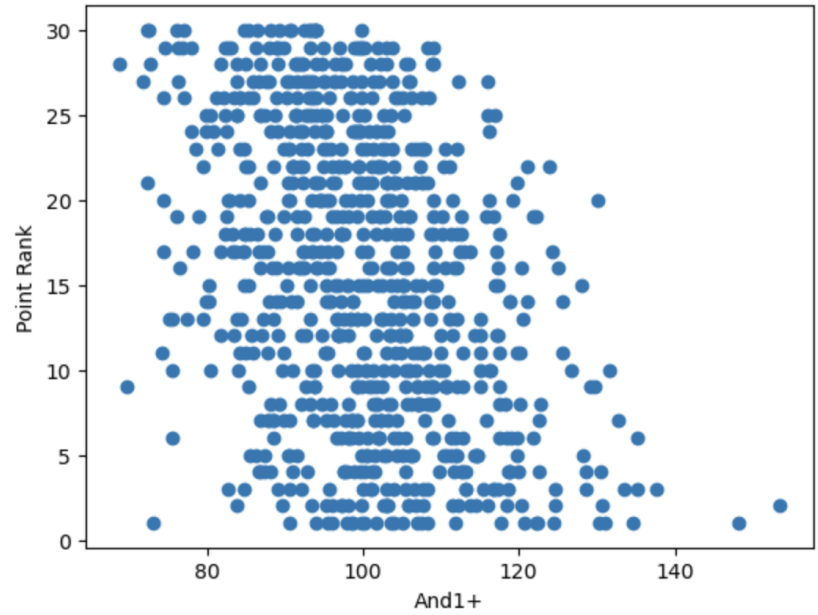
Now that we definitively know who the best and worst teams of all time are at drawing and-1 plays, we want to see if this has any true correlation to offensive and team success. In order to do this we created scatter plots and ran linear regression analyses comparing And1+ to the following variables: Win Percentage (Figure 25), Point Rank (where you finished in the points per game leaderboard at the end of the year) (Figure 26), Points Per Game (Figure 27), and Free Throws Attempted Per Game (Figure 28).

Unfortunately, none of our findings turned out to be significant. It turns out that And1+ did not heavily correlate to any of the dependent variables. The order of variables that correlated with And1+ from most to least was Free Throws Attempted Per Game (r-squared of 0.211), Point Rank (0.147), Points Per Game (0.043), and Win Percentage (0.002). So as you can see, the number of and-1 plays drawn by a team does not have much effect on offensive performance or overall team success.



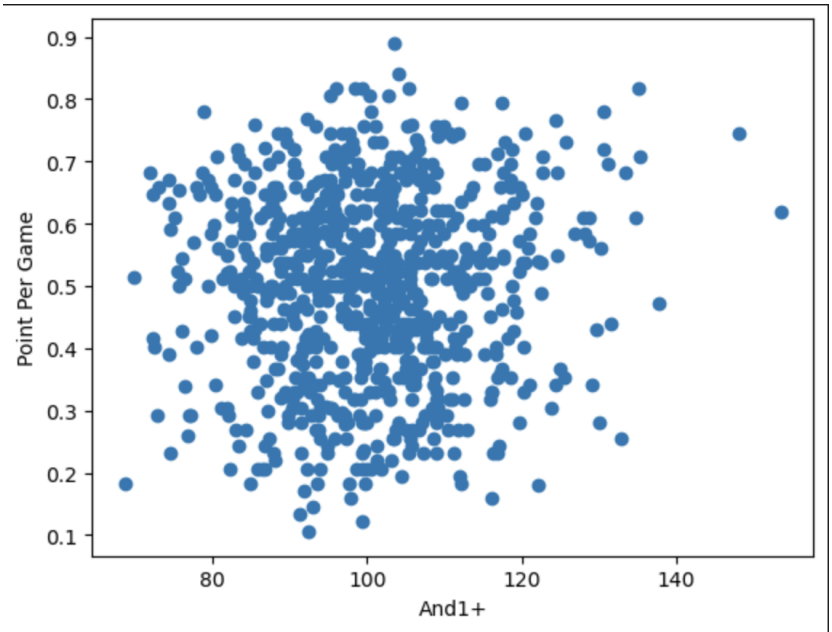

```
Slope: 0.0005320123779052339
Intercept: 0.4466004547768205
R-squared: 0.0018350963919279244
p-value: 0.22558127184311705
Standard Error: 0.00043868062185532905
```

Figure 25: Scatter plot and linear regression analysis of And1+ vs Win Percentage.




```
Slope: -0.2686594344904737
Intercept: 42.21135678165929
R-squared: 0.14713616484614112
p-value: 1.6413266224333804e-29
Standard Error: 0.022868476684424783
```

Figure 26: Scatter plot and linear regression analysis of And1+ vs Point Ranking.




```
Slope: 0.12537191662570746
Intercept: 88.38773348736275
R-squared: 0.04294897381456691
p-value: 3.136846622533044e-09
Standard Error: 0.020924082442706462
```

Figure 27: Scatter plot and linear regression analysis of And1+ vs Points Per Game.

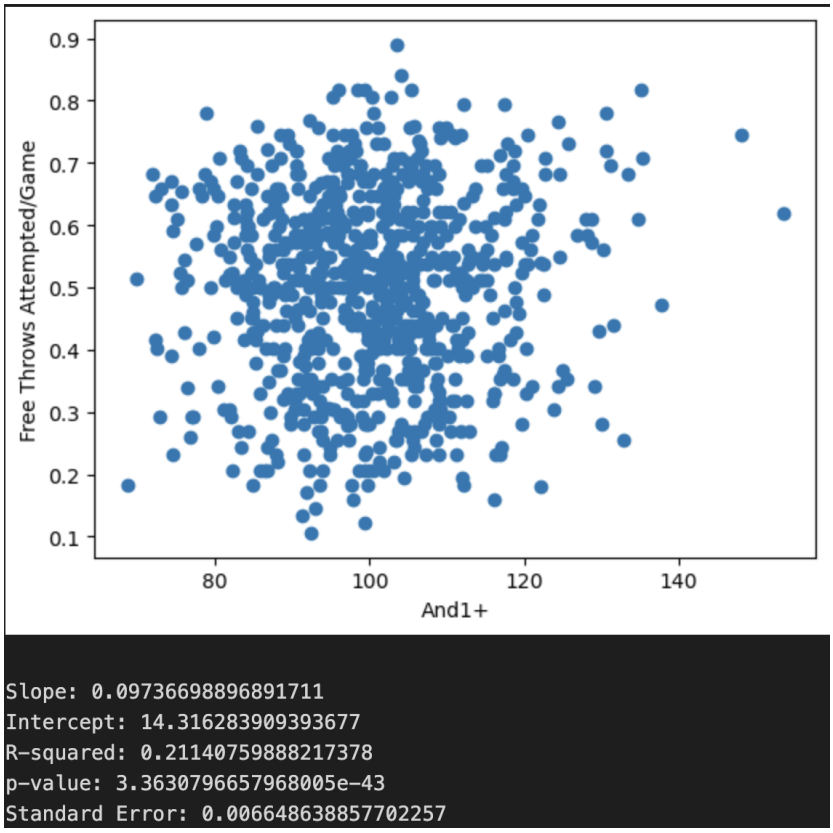


Figure 28: Scatter plot and linear regression analysis of And1+ vs Free Throws Attempted Per Game.

3.4 Defensive Team Performance

Now that we have looked at offensive team performance, it is time to look at defensive performance. Since teams can draw and-1 plays, the other team must be the one committing the and-1 fouls that result in and-1 plays. We wanted to look at how good teams were about committing and-1 fouls. We did this for both all time teams and teams in a single season.

3.4.1 All-Time

Just like drawing and-1 plays, teams that have committed the most and-1 fouls of all time really is not that significant when you think about it. However since we did it for offensive teams, we thought it would be cool to do the same for defensive teams and examine which franchises have committed the most and-1 fouls of all time on all shot types (Figure 29), 2pt shots (Figure 30), and 3pt shots (Figure 31). We found that the Golden State Warriors have committed the most and-1 fouls of all time for both all shot types and 2pt shots, followed by the Sacramento Kings, Dallas Mavericks, Washington Wizards, and New York Knicks. For 3pt shots, the Los Angeles Clippers have committed the most and-1 fouls of all time, followed by the Milwaukee Bucks, Miami Heat, Memphis Grizzlies, and Indiana Pacers.

Opposing_Team	And_1s	Shooting_Fouls	And_1s_2pt	And_1s_3pt
GSW	5004	22349	4894	110
SAC	4905	21128	4797	108
DAL	4838	21116	4754	84
WAS	4795	21496	4682	113
NYK	4773	21600	4680	93
PHX	4739	21096	4649	90
MIN	4683	21070	4586	97
BOS	4670	21866	4581	89
UTA	4669	21967	4575	94
TOR	4668	21138	4559	109
DEN	4659	21118	4573	86
MIL	4639	21229	4520	119
ORL	4633	20782	4529	104
LAC	4631	21583	4505	126
CHI	4612	20926	4501	111
PHI	4595	21332	4511	84
MIA	4593	20534	4475	118
ATL	4593	20706	4484	109
IND	4508	21330	4392	116
POR	4479	20352	4376	103

Figure 29: Top 20 teams that have committed the most and-1 fouls on all shot types in NBA history.

Opposing_Team	And_1s_2pt	Shooting_Fouls	And_1s	And_1s_3pt
GSW	4894	22349	5004	110
SAC	4797	21128	4905	108
DAL	4754	21116	4838	84
WAS	4682	21496	4795	113
NYK	4680	21600	4773	93
PHX	4649	21096	4739	90
MIN	4586	21070	4683	97
BOS	4581	21866	4670	89
UTA	4575	21967	4669	94
DEN	4573	21118	4659	86
TOR	4559	21138	4668	109
ORL	4529	20782	4633	104
MIL	4520	21229	4639	119
PHI	4511	21332	4595	84
LAC	4505	21583	4631	126
CHI	4501	20926	4612	111
ATL	4484	20706	4593	109
MIA	4475	20534	4593	118
IND	4392	21330	4508	116
POR	4376	20352	4479	103

Figure 30: Top 20 teams that have committed the most and-1 fouls on 2pt shots in NBA history.

Opposing_Team	And_1s_3pt	Shooting_Fouls	And_1s	And_1s_2pt
LAC	126	21583	4631	4505
MIL	119	21229	4639	4520
MIA	118	20534	4593	4475
MEM	116	17242	3813	3697
IND	116	21330	4508	4392
WAS	113	21496	4795	4682
CHI	111	20926	4612	4501
DET	110	20146	4438	4328
LAL	110	20666	4389	4279
GSW	110	22349	5004	4894
ATL	109	20706	4593	4484
TOR	109	21138	4668	4559
SAC	108	21128	4905	4797
ORL	104	20782	4633	4529
POR	103	20352	4479	4376
OKC	101	11473	2623	2522
CLE	99	20558	4422	4323
MIN	97	21070	4683	4586
UTA	94	21967	4669	4575
NYK	93	21600	4773	4680

Figure 31: Top 20 teams that have committed the most and-1 fouls on 3pt shots in NBA history.

3.4.2 Single-Season

Now to get a little more specific than all time rankings, we wanted to see which individual teams committed the most and-1 fouls in a single season. We calculated the single season team with the most and-1 fouls on all shot types (Figure 32), the least and-1 fouls on all shot types (Figure 33), the most and-1 fouls on 2pt shots (Figure 34), the least and-1 fouls on 2pt shots (Figure 35), the most and-1 fouls on 3pt shots (Figure 36), and the least and-1 fouls on 3pt shots (Figure 37).

When it comes to the most and-1 fouls committed, the 2008-2009 Kings, 2009-2010 Warriors, 2006-2007 Warriors, 2018-2019 Knicks, and 2022-2023 Wizards are at the top. Whereas the fewest and-1 fouls committed came from the 1998-1999 Hawks, 1998-1999 76ers, 1998-1999 Magic, 1998-1999 Pacers, and 1998-1999 Heat. When it comes to 3pt shots, the teams that committed the most and-1 fouls were the 2019-2020 Bucks, 2018-2019 Clippers, 2018-2019 Pistons, 2021-2022 Wizards, and 2019-2020 Trail Blazers. On the opposite side, there were once again over 50 teams that went an entire season without committing an and-1 fouls on a 3pt shot.

Opposing_Team	Season	And_1s	Shooting_Fouls	And_1s_2pt	And_1s_3pt
SAC	2008-09	259	1039	256	3
GSW	2009-10	245	1012	244	1
GSW	2006-07	245	1027	239	6
NYK	2018-19	242	868	230	12
WAS	2022-23	241	862	234	7
ATL	2005-06	240	970	239	1
DET	2022-23	239	914	229	10
NOP	2022-23	236	886	228	8
BKN	2022-23	236	890	231	5
SAC	2022-23	234	851	223	11
OKC	2022-23	234	919	226	8
SAS	2022-23	232	938	226	6
ORL	2022-23	231	847	221	10
PHX	2022-23	230	924	224	6
GSW	2010-11	230	939	228	2
HOU	2021-22	229	828	221	8
MIL	2006-07	228	869	224	4
NYK	2022-23	227	852	219	8
CHA	2022-23	225	852	219	6
SAC	2009-10	225	905	222	3

Figure 32: Top 20 single-season teams with the most and-1s committed on all shot types in NBA history.

Opposing_Team	Season	And_1s	Shooting_Fouls	And_1s_2pt	And_1s_3pt
ATL	1998-99	66	439	66	0
PHI	1998-99	77	484	77	0
ORL	1998-99	80	397	79	1
IND	1998-99	80	490	80	0
MIA	1998-99	82	445	81	1
TOR	1998-99	82	507	81	1
HOU	1998-99	83	416	82	1
SAS	1998-99	85	476	85	0
NYK	1998-99	88	517	86	2
PHX	1998-99	90	476	90	0
DET	1998-99	90	501	87	3
MIL	1998-99	90	535	88	2
CLE	1998-99	91	490	89	2
POR	1998-99	97	486	97	0
UTA	1998-99	98	513	98	0
LAL	1998-99	99	545	97	2
LAL	2011-12	104	499	104	0
NJN	1998-99	106	508	105	1
GSW	1998-99	106	534	106	0
SEA	1998-99	107	448	107	0

Figure 33: Top 20 single-season teams with the least and-1s committed on all shot types in NBA history.

Opposing_Team	Season	And_1s_2pt	Shooting_Fouls	And_1s	And_1s_3pt
SAC	2008-09	256	1039	259	3
GSW	2009-10	244	1012	245	1
ATL	2005-06	239	970	240	1
GSW	2006-07	239	1027	245	6
WAS	2022-23	234	862	241	7
BKN	2022-23	231	890	236	5
NYK	2018-19	230	868	242	12
DET	2022-23	229	914	239	10
NOP	2022-23	228	886	236	8
GSW	2010-11	228	939	230	2
OKC	2022-23	226	919	234	8
SAS	2022-23	226	938	232	6
MIL	2006-07	224	869	228	4
PHX	2022-23	224	924	230	6
SAC	2022-23	223	851	234	11
SAC	2009-10	222	905	225	3
HOU	2021-22	221	828	229	8
ORL	2022-23	221	847	231	10
DAL	2006-07	221	940	221	0
BOS	2005-06	221	1000	223	2

Figure 34: Top 20 single-season teams with the most and-1s committed on 2pt shots in NBA history.

Opposing_Team	Season	And_1s_2pt	Shooting_Fouls	And_1s	And_1s_3pt
ATL	1998-99	66	439	66	0
PHI	1998-99	77	484	77	0
ORL	1998-99	79	397	80	1
IND	1998-99	80	490	80	0
MIA	1998-99	81	445	82	1
TOR	1998-99	81	507	82	1
HOU	1998-99	82	416	83	1
SAS	1998-99	85	476	85	0
NYK	1998-99	86	517	88	2
DET	1998-99	87	501	90	3
MIL	1998-99	88	535	90	2
CLE	1998-99	89	490	91	2
PHX	1998-99	90	476	90	0
POR	1998-99	97	486	97	0
LAL	1998-99	97	545	99	2
UTA	1998-99	98	513	98	0
ORL	2011-12	103	545	107	4
LAL	2011-12	104	499	104	0
NJN	1998-99	105	508	106	1
GSW	1998-99	106	534	106	0

Figure 35: Top 20 single-season teams with the least and-1s committed on 2pt shots in NBA history.

Opposing_Team	Season	And_1s_3pt	Shooting_Fouls	And_1s	And_1s_2pt
MIL	2019-20	20	641	144	124
LAC	2018-19	19	851	215	196
DET	2018-19	18	771	206	188
WAS	2021-22	18	803	219	201
POR	2019-20	16	741	178	162
MIA	2019-20	16	745	195	179
OKC	2021-22	16	770	214	198
SAC	2018-19	16	836	209	193
CHI	2017-18	15	704	198	183
PHX	2019-20	15	751	212	197
WAS	2017-18	15	796	215	200
CLE	2018-19	14	724	206	192
HOU	2019-20	14	734	187	173
LAC	2019-20	14	747	176	162
GSW	2019-20	13	595	152	139
ORL	2019-20	13	613	165	152
CHA	2017-18	13	640	176	163
TOR	2019-20	13	674	178	165
IND	2018-19	13	734	186	173
OKC	2018-19	13	764	198	185

Figure 36: Top 20 single-season teams with the most and-1s committed on 3pt shots in NBA history.

Opposing_Team	Season	And_1s_3pt	Shooting_Fouls	And_1s	And_1s_2pt
ATL	1998-99	0	439	66	66
SEA	1998-99	0	448	107	107
SAC	1998-99	0	458	109	109
PHX	1998-99	0	476	90	90
SAS	1998-99	0	476	85	85
DAL	1998-99	0	480	113	113
PHI	1998-99	0	484	77	77
POR	1998-99	0	486	97	97
IND	1998-99	0	490	80	80
LAL	2011-12	0	499	104	104
UTA	1998-99	0	513	98	98
GSW	1998-99	0	534	106	106
PHX	2011-12	0	572	134	134
HOU	2011-12	0	580	123	123
LAC	1998-99	0	596	129	129
NJN	2011-12	0	621	154	154
SAS	2001-02	0	670	140	140
HOU	2001-02	0	684	149	149
PHI	2001-02	0	688	131	131
DAL	2015-16	0	690	165	165

Figure 37: Top 20 single-season teams with the least and-1s committed on 3pt shots in NBA history.

Once again, the difference in number of games for a season and difference in eras contributed to the single season leaders in and-1 fouls committed. So we once again had to calculate our And1+ stat to determine who were officially the best and worst teams about committing and-1 fouls. However this time, we had to reverse the meaning of the statistic and calculate defensive And1+. This meant that while 100 was still average compared to the rest of the league, an And1+ of 105 meant that you committed 5% less and-1 fouls than the average team that season, and an And1+ of 95 meant that you committed 5% more and-1 fouls than the average team that season. So we calculated who had the best defensive And1+ (Figure 38), and the worst defensive And1+ (Figure 39) using this formula: $def\ and1+ = (\frac{league\ avg\ and1\ fouls\ that\ season}{and1\ fouls\ that\ season}) * 100$.

We found that the team with the best defensive And1+ of all time was still the 1998-1999 Atlanta Hawks with a defensive And1+ of 150, meaning they committed 50% less and-1 fouls than the average team that season. They were followed by the 2015-2016 Spurs, 2020-2021 Hornets, 2014-2015 Cavaliers, and 2009-2010 Lakers. The worst defensive And1+ of all time also still came from the 2008-2009 Kings with 72.02, meaning they committed 27.98% more and-1 fouls than the average team that season. They were followed by the 1998-1999 Clippers, 1998-1999 Grizzlies, 2005-2006 Hawks, and 2009-2010 Suns.

Opposing_Team	Season	And_1+	And_1s	Shooting_Fouls
ATL	1998-99	150.00	66	439
SAS	2015-16	138.36	120	654
CHA	2020-21	137.86	118	600
CLE	2014-15	134.66	116	675
LAL	2009-10	133.68	144	714
SAS	2007-08	129.44	142	695
DET	2004-05	129.15	137	701
PHI	1998-99	128.57	77	484
CLE	2019-20	127.95	130	532
HOU	2008-09	127.76	146	752
HOU	2005-06	127.55	147	835
POR	2010-11	127.01	145	733
NOH	2008-09	126.89	147	744
DET	2003-04	126.19	124	688
SAS	2002-03	126.09	127	688
SAS	2000-01	125.85	125	691
POR	2003-04	125.18	125	648
NOH	2003-04	124.19	126	712
DET	2005-06	124.17	151	704
ORL	1998-99	123.75	80	397

Figure 38: Top 20 single-season teams with the highest defensive and1+ in NBA history.

Opposing_Team	Season	And_1+	And_1s	Shooting_Fouls
SAC	2008-09	72.02	259	1039
LAC	1998-99	76.74	129	596
VAN	1998-99	77.95	127	547
ATL	2005-06	78.12	240	970
PHX	2019-20	78.46	212	751
GSW	2009-10	78.57	245	1012
DAL	2012-13	78.59	197	851
NYK	2018-19	79.31	242	868
MIN	2000-01	79.45	198	834
GSW	2010-11	80.07	230	939
CLE	2012-13	80.22	193	808
WAS	2019-20	80.35	207	774
SAC	1996-97	80.47	215	862
UTA	2011-12	80.74	158	711
CHH	1998-99	81.15	122	522
CHA	2011-12	81.78	156	643
GSW	2006-07	81.80	245	1027
PHI	2013-14	81.98	206	921
HOU	2021-22	82.10	229	828
GSW	1999-00	82.48	200	884

Figure 39: Top 20 single-season teams with the lowest defensive and1+ in NBA history.

3.4.3 Correlation to Team Success

Now that we definitively know who the best and worst teams of all time are at committing and-1 fouls, we want to see if this has any true correlation to defensive and team success. In order to do this we created scatter plots and ran linear regression analyses comparing defensive And1+ to the following variables: Win Percentage (Figure 40) and Personal Fouls Per Game (Figure 41).

Unfortunately, none of our findings turned out to be significant. It turns out that defensive And1+ did not heavily correlate to any of the dependent variables. The order of variables that correlated with And1+ from most to least was Win Percentage (r-squared of 0.140) and Personal Fouls Per Game (0.124). So as you can see, the number of and-1 fouls committed by a team does not have much effect on defensive performance or overall team success. However, it does appear that limiting the number of and-1 fouls you commit has more of an effect on team success than drawing and-1 plays.

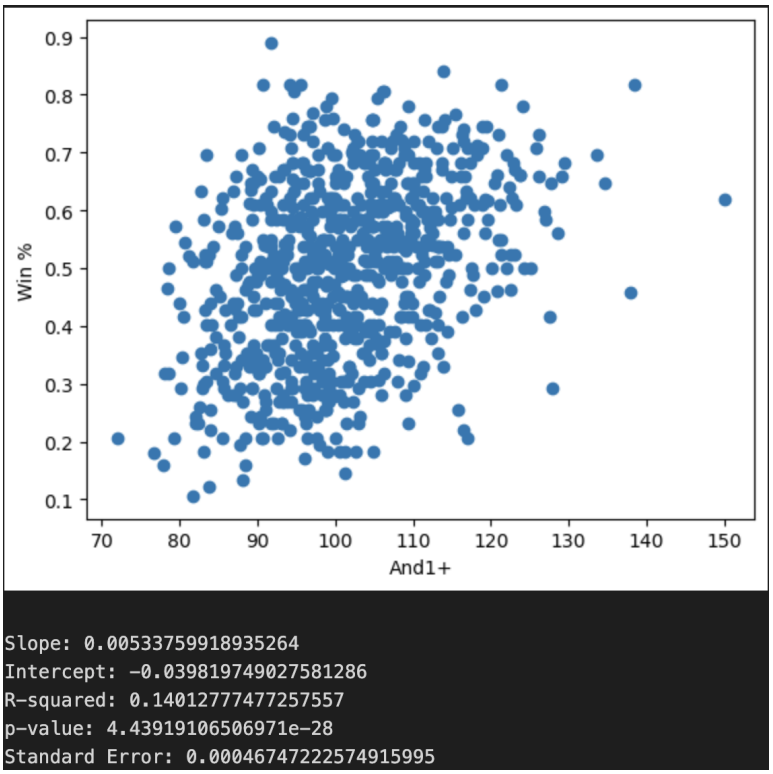


Figure 40: Scatter plot and linear regression analysis of Defensive And1+ vs Win Percentage.

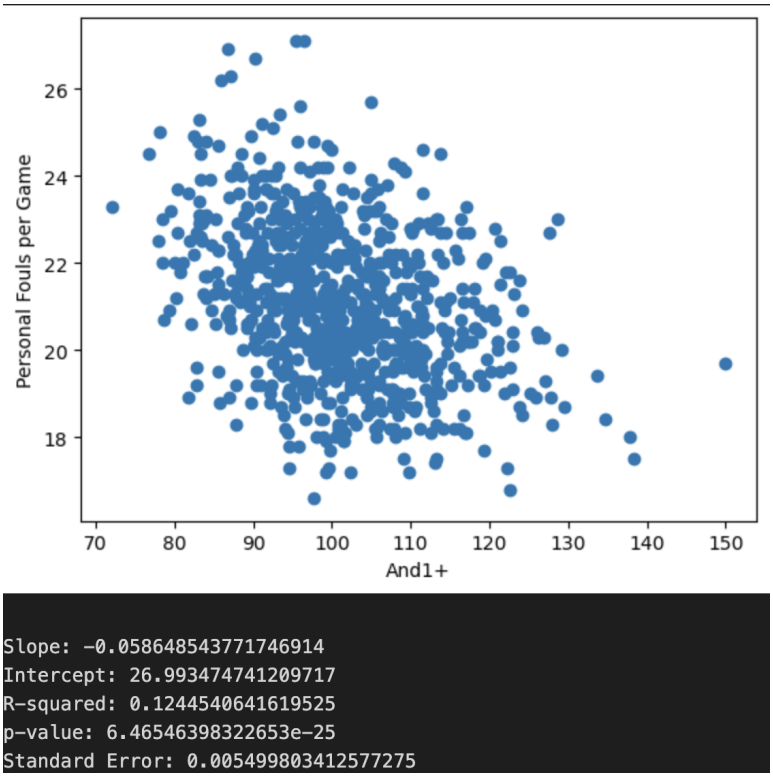


Figure 41: Scatter plot and linear regression analysis of Defensive And1+ vs Personal Fouls Committed Per Game.

References

[1] Garnepudi, Vineeth, and Jason Ganslaw. "Group1-NBA-API-4-4-23_Documentation2-V2." 4 Apr. 2023.

This group 1's documentation. It gives details on the folder structure and information on many of the folders in the NBA API. We used this to navigate the NBA API GitHub so we could retrieve all the necessary information we needed for our project.

[2] Garnpudi, Vineeth, and Jason Ganslaw. "Group1-NBA-API-28-3-23_Documentation." 28 Mar. 2023.

This is group 1's documentation. It gives lots of background information on the NBA API which we used to learn more about the technology. It also gives information on the static resources such as teams and players, which we used to retrieve team and player data from the API in our project.

[3] Guy, Eric, and Ethan Posner. "NBA_API 4_20 Group 4 Documentation ." 20 Apr. 2023.

This is group 4's documentation. This documentation was used when we needed a way to create a retry wrapper for our large datasets. We needed strategies to get our program to stop timing out when accessing the NBA API, and this documentation provided us with the strategies and code to do that.

[4] McCoy, Scott. Murach's Python for Data Analysis. Mike Murach amp; Associates, Inc, 2021.

This book describes many strategies and skills needed to analyze data using the Python programming language. We used this book in conjunction with the NBA API to gather and analyze the data for our project.

[5] Patel, Swar. "Swar/NBA_API: An API Client Package to Access the Apis for Nba.Com." GitHub-NBA API, github.com/swar/nba_api. Accessed 17 May 2023.

This is the API documentation GitHub that we used to retrieve data for our project. It is filled with all of the data provided by the official NBA API. It explains how to acces the various endpoints and statistics to get data about players, teams, games and seasons through the NBA.

[6] Posner, Ethan, and Eric Guy. "Group4-NBA-API-30-3-23_Documentation-V2." 30 Mar. 2023.

This is Group 4's documentation. It outlines the use of LeagueGameFinder, PlayByPlayV2, and BoxScore endpoints. We used all of these endpoints in our project and constantly referred to this documentation in order to get information about play-by-play data and specific games.