assignment2

December 14, 2023

[1]:
```python
# Initialize Otter
import otter
grader = otter.Notebook("assignment2.ipynb")
```

1 Assignment 2: Exploratory Data Analysis in Professional Basketball

In this assignment we'll conduct an exploratory data analysis of professional basketball data. Basketball is a team sport in which the goal is to try to outscore the amount in a fixed amount of time. Points are scored (either 2 or 3 points) by putting the ball throw a hoop on one end of the court. An attempt at putting the ball throw the hoop is known as a "shot". If helpful, you can read more about the rules of basketball.

The National Basketball Association (NBA) is the professional basketball league in the United States and provides a nice website with many statistics gathered on teams and players in the league: `http://stat.nba.com`.

1.1 Question 1: Managing data files

We will use data that is available from NBA. Although NBA doesn't officially make the data API (application programming interface) public, people have figured out ways to access their data programmatically (1, 2). However, NBA does not offer an official API and it is possible to get our JupyterHub blocked by the site if we use them. Therefore, in this assignment, the raw data downloads are provided to you in a zip file: https://ucsb.box.com/shared/static/z6y3etgikbzbnf0ld4brvc95xtgjcrie.zip

Download and unzip the file to a directory named `data` using command line commands (unzipping on Windows and Mac may not work because different OS have different constraints on filename lengths, etc.). Adding an exclamation point in the Jupyter notebook cell indicates that `bash` shell interpreter will execute your command.

```bash
wget -nc https://ucsb.box.com/shared/static/z6y3etgikbzbnf0ld4brvc95xtgjcrie.zip -O nba-data.zi
unzip -o nba-data.zip -d data
```

What these commands are doing: * `wget` downloads files (what do each of the pieces do?) * `unzip` will unzip `nba-data.zip` into a directory named `data` (specified by `-d data`) and will overwrite any same filenames when extracting (specified by `-o`).

[2]:
```
# Run your commands in this cell
#!wget -nc https://ucsb.box.com/shared/static/z6y3etgikbzbnf0ld4brvc95xtgjcrie.
↪zip -O nba-data.zip
#!unzip -o nba-data.zip -d data
```

After unzipping the files, you will find three types of files in `data/` directory:

- Team data: `commonTeamYears?LeagueID=00&Season=2018-19`
- Player data: `commonallplayers?LeagueID=00&Season=2018-19&IsOnlyCurrentSeason=0`
- Player's shot data: `shotchartdetail?PlayerID=[PlayerID]&PlayerPosition=&Season=2018-19&Context`

Each player's shot data is identified by replacing `[PlayerID]` with their numeric ID.

Here is how we will read in the data: * Each data file contains text in JSON (Javascript Object Notation) format. * First, read the data content as text (using Path.read_text() from `pathlib` module) * Second, we convert it to a Python dictionary format (using json.loads() in `json` module) * Third, identify DataFrame content * Fourth, identify DataFrame header * Fifth, assemble DataFrame

Another way to unzip a file is using `zipfile`.

[3]:
```
#import zipfile
#with zipfile.ZipFile('nba-data.zip', 'r') as zip_ref:
#    zip_ref.extractall('data')
```

1.1.1 Question 1a: Team Data

Read team data file into a pandas data frame named `allteams` starting from the given code below.

[4]:
```
from pathlib import Path
import json
import pandas as pd
import numpy as np

fname  = 'data/commonTeamYears?LeagueID=00&Season=2018-19' # directory_name/
↪file_name
step_1 = Path(fname).read_text()                 # str
step_2 = json.loads(step_1)                      # dict
step_3 = step_2['resultSets'][0]['rowSet']       # list
step_4 = step_2['resultSets'][0]['headers']      # list
```

[5]:
```
# print out each of step_1 through step_4 and understand what each line does
```

[6]:
```
#print(step_1)
#print(step_2)
print(step_3)
#print(step_4)
```

```
[['00', 1610612737, '1949', '2019', 'ATL'], ['00', 1610612738, '1946', '2019',
'BOS'], ['00', 1610612740, '2002', '2019', 'NOP'], ['00', 1610612741, '1966',
```

```
'2019', 'CHI'], ['00', 1610612742, '1980', '2019', 'DAL'], ['00', 1610612743,
'1976', '2019', 'DEN'], ['00', 1610612745, '1967', '2019', 'HOU'], ['00',
1610612746, '1970', '2019', 'LAC'], ['00', 1610612747, '1948', '2019', 'LAL'],
['00', 1610612748, '1988', '2019', 'MIA'], ['00', 1610612749, '1968', '2019',
'MIL'], ['00', 1610612750, '1989', '2019', 'MIN'], ['00', 1610612751, '1976',
'2019', 'BKN'], ['00', 1610612752, '1946', '2019', 'NYK'], ['00', 1610612753,
'1989', '2019', 'ORL'], ['00', 1610612754, '1976', '2019', 'IND'], ['00',
1610612755, '1949', '2019', 'PHI'], ['00', 1610612756, '1968', '2019', 'PHX'],
['00', 1610612757, '1970', '2019', 'POR'], ['00', 1610612758, '1948', '2019',
'SAC'], ['00', 1610612759, '1976', '2019', 'SAS'], ['00', 1610612760, '1967',
'2019', 'OKC'], ['00', 1610612761, '1995', '2019', 'TOR'], ['00', 1610612762,
'1974', '2019', 'UTA'], ['00', 1610612763, '1995', '2019', 'MEM'], ['00',
1610612764, '1961', '2019', 'WAS'], ['00', 1610612765, '1948', '2019', 'DET'],
['00', 1610612766, '1988', '2019', 'CHA'], ['00', 1610612739, '1970', '2019',
'CLE'], ['00', 1610612744, '1946', '2019', 'GSW'], ['00', 1610610031, '1946',
'1946', None], ['00', 1610610029, '1948', '1948', None], ['00', 1610610025,
'1946', '1949', None], ['00', 1610610034, '1946', '1949', None], ['00',
1610610036, '1946', '1950', None], ['00', 1610610024, '1947', '1954', None],
['00', 1610610027, '1949', '1949', None], ['00', 1610610030, '1949', '1952',
None], ['00', 1610610033, '1949', '1949', None], ['00', 1610610037, '1949',
'1949', None], ['00', 1610610023, '1949', '1949', None], ['00', 1610610026,
'1946', '1946', None], ['00', 1610610028, '1946', '1946', None], ['00',
1610610032, '1946', '1948', None], ['00', 1610610035, '1946', '1946', None]]
```

Use variables constructed above to assemble `allteams` DataFrame.

Drop any teams that no longer exist as of 2019. These teams show None in `ABBREVIATION` column.

[7]:
```
allteams = pd.DataFrame(data = step_3, columns = step_4)
allteams = allteams[allteams["MAX_YEAR"] >= "2019"]
print(allteams)
```

```
   LEAGUE_ID     TEAM_ID MIN_YEAR MAX_YEAR ABBREVIATION
0         00  1610612737     1949     2019          ATL
1         00  1610612738     1946     2019          BOS
2         00  1610612740     2002     2019          NOP
3         00  1610612741     1966     2019          CHI
4         00  1610612742     1980     2019          DAL
5         00  1610612743     1976     2019          DEN
6         00  1610612745     1967     2019          HOU
7         00  1610612746     1970     2019          LAC
8         00  1610612747     1948     2019          LAL
9         00  1610612748     1988     2019          MIA
10        00  1610612749     1968     2019          MIL
11        00  1610612750     1989     2019          MIN
12        00  1610612751     1976     2019          BKN
13        00  1610612752     1946     2019          NYK
14        00  1610612753     1989     2019          ORL
15        00  1610612754     1976     2019          IND
```

```
16         00  1610612755     1949     2019          PHI
17         00  1610612756     1968     2019          PHX
18         00  1610612757     1970     2019          POR
19         00  1610612758     1948     2019          SAC
20         00  1610612759     1976     2019          SAS
21         00  1610612760     1967     2019          OKC
22         00  1610612761     1995     2019          TOR
23         00  1610612762     1974     2019          UTA
24         00  1610612763     1995     2019          MEM
25         00  1610612764     1961     2019          WAS
26         00  1610612765     1948     2019          DET
27         00  1610612766     1988     2019          CHA
28         00  1610612739     1970     2019          CLE
29         00  1610612744     1946     2019          GSW
```

[8]:
```
grader.check("q1a")
```

[8]:
```
q1a results: All test cases passed!
```

1.1.2 Question 1b: Player Data

`pathlib` has flexible ways to specify file and directory paths. For example, the following are equivalent:

- `Path('data/commonallplayers?LeagueID=00&Season=2018-19&IsOnlyCurrentSeason=0')`
- `Path('data') / 'commonallplayers?LeagueID=00&Season=2018-19&IsOnlyCurrentSeason=0')`
- `Path('data').joinpath('commonallplayers?LeagueID=00&Season=2018-19&IsOnlyCurrentSeason=0')`

Read players data file with name `data/commonallplayers?LeagueID=00&Season=2018-19&IsOnlyCurrentSeason` Assemble pandas DataFrame with name `allplayers`. Set row index to be `PERSON_ID` and `sort_index`.

[9]:
```
dirname  = 'data' # directory_name
filename = 'commonallplayers?LeagueID=00&Season=2018-19&IsOnlyCurrentSeason=0'␣
 ↪# file_name
step_1 = Path(dirname).joinpath(filename).read_text()
step_2 = json.loads(step_1)
step_3 = step_2['resultSets'][0]['rowSet']
step_4 = step_2['resultSets'][0]['headers']

allplayers = pd.DataFrame(data = step_3, columns = step_4).
 ↪set_index("PERSON_ID").sort_index()
print(allplayers)
```

```
          DISPLAY_LAST_COMMA_FIRST DISPLAY_FIRST_LAST  ROSTERSTATUS FROM_YEAR  \
PERSON_ID
2                     Scott, Byron        Byron Scott             0      1983
3                      Long, Grant         Grant Long             0      1988
7                     Schayes, Dan        Dan Schayes             0      1981
```

```
9                 Threatt, Sedale     Sedale Threatt             0       1983
12                    King, Chris         Chris King             0       1993
…                           …                 …           …        …
1629956              Brown, Barry        Barry Brown             0       2019
1629962            Cannady, Devin      Devin Cannady             0       2019
1629967           Flatten, Skyler     Skyler Flatten             0       2019
1630001              Morgan, Matt        Matt Morgan             0       2019
1630003               Scott, MIke         MIke Scott             0       2019

          TO_YEAR      PLAYERCODE  TEAM_ID TEAM_CITY TEAM_NAME  \
PERSON_ID
2            1996     byron_scott        0
3            2002      grant_long        0
7            1998     dan_schayes        0
9            1996  sedale_threatt        0
12           1998      chris_king        0
…           …            …      …        …        …
1629956      2019     barry_brown        0
1629962      2019   devin_cannady        0
1629967      2019  skyler_flatten        0
1630001      2019     matt_morgan        0
1630003      2019  tmp_mike_scott        0

          TEAM_ABBREVIATION TEAM_CODE GAMES_PLAYED_FLAG  \
PERSON_ID
2                                                   Y
3                                                   Y
7                                                   Y
9                                                   Y
12                                                  Y
…                       …       …               …
1629956                                             N
1629962                                             N
1629967                                             N
1630001                                             N
1630003                                             N

          OTHERLEAGUE_EXPERIENCE_CH
PERSON_ID
2                                00
3                                00
7                                00
9                                00
12                               00
…                               …
1629956                          00
1629962                          00
1629967                          00
```

```
1630001                          00
1630003                          00

[4540 rows x 13 columns]
```

[10]:
```
grader.check("q1b")
```

[10]:
```
q1b results: All test cases passed!
```

1.1.3 Question 1c: Shots Data

`pathlib` can also find all filenames that match a given pattern using `Path.glob()` method.

For example, teams data and players data start with the pattern `common` followed by a wildcard `*`: `common*`.

We can use this to retrieve two file names with one call:

[11]:
```
two_files = Path('data').glob('common*') # generator: https://www.educative.io/
↪edpresso/generator-vs-iterator-in-python
list(two_files)                          # list
```

[11]:
```
[PosixPath('data/commonTeamYears?LeagueID=00&Season=2018-19'),
 PosixPath('data/commonallplayers?LeagueID=00&Season=2018-
19&IsOnlyCurrentSeason=0')]
```

All file names for shots data start with `shotchartdetail`.

Use this as the pattern to * First, read all file names into `allshots_files` * Second, loop over each file in `allshots_files` and assemble a dataframe * Third, add as an element in a list named `allshots_list` (each file is an data frame item in the list). * Fourth, concatenate all dataframes into one dataframe named `allshots`. Set the row index to be `PLAYER_ID` and `sort_index`.

[12]:
```
allshots_files = list(Path('data').glob('shotchartdetail*'))
allshots_files.sort()
allshots_list  = list()

for f in allshots_files:
    step_1 = f.read_text()
    step_2 = json.loads(step_1)
    step_3 = step_2['resultSets'][0]['rowSet']
    step_4 = step_2['resultSets'][0]['headers']
    allshots_list.append(pd.DataFrame(data = step_3, columns = step_4))

allshots = pd.concat(allshots_list).set_index("PLAYER_ID").sort_index()
print(allshots)
```

```
                   GRID_TYPE     GAME_ID GAME_EVENT_ID     PLAYER_NAME  \
PLAYER_ID
1713       Shot Chart Detail  0021800007             9    Vince Carter
```

```
1713       Shot Chart Detail  0021800928          551    Vince Carter
1713       Shot Chart Detail  0021800928          417    Vince Carter
1713       Shot Chart Detail  0021800928          278    Vince Carter
1713       Shot Chart Detail  0021800928          107    Vince Carter
…                       …        …          …             …
1629541    Shot Chart Detail  0021801147          193  Dairis Bertans
1629541    Shot Chart Detail  0021801147          208  Dairis Bertans
1629541    Shot Chart Detail  0021801147          224  Dairis Bertans
1629541    Shot Chart Detail  0021801147          244  Dairis Bertans
1629541    Shot Chart Detail  0021801215          411  Dairis Bertans

              TEAM_ID             TEAM_NAME PERIOD MINUTES_REMAINING  \
PLAYER_ID
1713       1610612737         Atlanta Hawks      1                11
1713       1610612737         Atlanta Hawks      4                 9
1713       1610612737         Atlanta Hawks      3                 6
1713       1610612737         Atlanta Hawks      2                 4
1713       1610612737         Atlanta Hawks      1                 3
…               …                     …    …               …
1629541    1610612740  New Orleans Pelicans      2                10
1629541    1610612740  New Orleans Pelicans      2                 9
1629541    1610612740  New Orleans Pelicans      2                 8
1629541    1610612740  New Orleans Pelicans      2                 7
1629541    1610612740  New Orleans Pelicans      3                 6

          SECONDS_REMAINING   EVENT_TYPE  …         SHOT_ZONE_AREA  \
PLAYER_ID                                 …
1713                     44  Missed Shot  …              Center(C)
1713                     15    Made Shot  …              Center(C)
1713                     51    Made Shot  …  Right Side Center(RC)
1713                     16  Missed Shot  …              Center(C)
1713                     24    Made Shot  …          Right Side(R)
…                      …          …   …                    …
1629541                  16    Made Shot  …   Left Side Center(LC)
1629541                  30  Missed Shot  …              Center(C)
1629541                  48  Missed Shot  …   Left Side Center(LC)
1629541                  34  Missed Shot  …  Right Side Center(RC)
1629541                   6  Missed Shot  …          Right Side(R)

           SHOT_ZONE_RANGE SHOT_DISTANCE LOC_X LOC_Y SHOT_ATTEMPTED_FLAG  \
PLAYER_ID
1713               24+ ft.            27    74   266                   1
1713       Less Than 8 ft.             0     2     7                   1
1713               24+ ft.            24   131   211                   1
1713       Less Than 8 ft.             6   -58    34                   1
1713              8-16 ft.             9    90    30                   1
…                     …            …    …    …                  …
1629541            24+ ft.            26  -190   184                   1
```

```
1629541              24+ ft.              25    -22    258                 1
1629541              24+ ft.              25   -149    211                 1
1629541              24+ ft.              25    185    174                 1
1629541              24+ ft.              22    226      6                 1

          SHOT_MADE_FLAG GAME_DATE  HTM  VTM
PLAYER_ID
1713                   0  20181017  NYK  ATL
1713                   1  20190301  ATL  CHI
1713                   1  20190301  ATL  CHI
1713                   0  20190301  ATL  CHI
1713                   1  20190301  ATL  CHI
…                    …         …    …    …
1629541                1  20190331  NOP  LAL
1629541                0  20190331  NOP  LAL
1629541                0  20190331  NOP  LAL
1629541                0  20190331  NOP  LAL
1629541                0  20190409  NOP  GSW

[217317 rows x 23 columns]
```

[13]:
```
grader.check("q1c")
```

[13]:
```
q1c results: All test cases passed!
```

1.1.4 Question 1d: Extract Stephen Curry's Shot Data

Use `allplayers.query()` to find the player id (index) associated with the player named "Stephen Curry". Set the value of `PlayerID` as `curry_id` of type `str`.

Subset all of Stephen Curry's shots in a data frame named `curry_data`. Also, set the dtype of `SHOT_MADE_FLAG` to `'bool'` in one command. Something like:

`curry_data = allshots.query(???).astype(????)`

[14]:
```
# fill-in all ...
query_str = 'DISPLAY_FIRST_LAST == "Stephen Curry"'
curry_id = str(allplayers.query(query_str).index.values[0])
curry_data = allshots.query('PLAYER_ID == ' + curry_id).
 ↪astype({'SHOT_MADE_FLAG' : 'bool'})
#curry_data = curry_data.set_index("PLAYER_ID")
print(curry_data)
```

```
                   GRID_TYPE     GAME_ID GAME_EVENT_ID    PLAYER_NAME  \
PLAYER_ID
201939     Shot Chart Detail  0021800862           117  Stephen Curry
201939     Shot Chart Detail  0021800862           600  Stephen Curry
201939     Shot Chart Detail  0021800862           576  Stephen Curry
201939     Shot Chart Detail  0021800862           484  Stephen Curry
```

```
201939     Shot Chart Detail  0021800862          467  Stephen Curry
…                       …         …           …          …
201939     Shot Chart Detail  0021800494          563  Stephen Curry
201939     Shot Chart Detail  0021800494          510  Stephen Curry
201939     Shot Chart Detail  0021800494          467  Stephen Curry
201939     Shot Chart Detail  0021800494          447  Stephen Curry
201939     Shot Chart Detail  0021800494          685  Stephen Curry

              TEAM_ID             TEAM_NAME PERIOD MINUTES_REMAINING  \
PLAYER_ID
201939     1610612744  Golden State Warriors      1                 3
201939     1610612744  Golden State Warriors      4                 5
201939     1610612744  Golden State Warriors      4                 6
201939     1610612744  Golden State Warriors      3                 2
201939     1610612744  Golden State Warriors      3                 3
…                 …                      …     …               …
201939     1610612744  Golden State Warriors      4                10
201939     1610612744  Golden State Warriors      3                 2
201939     1610612744  Golden State Warriors      3                 5
201939     1610612744  Golden State Warriors      3                 6
201939     1610612744  Golden State Warriors      4                 2

          SECONDS_REMAINING   EVENT_TYPE  …         SHOT_ZONE_AREA  \
PLAYER_ID                                 …
201939                   55    Made Shot  …              Center(C)
201939                    9  Missed Shot  …  Right Side Center(RC)
201939                   55  Missed Shot  …           Right Side(R)
201939                   24  Missed Shot  …            Left Side(L)
201939                    3  Missed Shot  …           Right Side(R)
…                       …           …  …                      …
201939                   50  Missed Shot  …  Right Side Center(RC)
201939                   25  Missed Shot  …   Left Side Center(LC)
201939                   33  Missed Shot  …              Center(C)
201939                   47    Made Shot  …            Left Side(L)
201939                    6  Missed Shot  …  Right Side Center(RC)

           SHOT_ZONE_RANGE SHOT_DISTANCE LOC_X LOC_Y SHOT_ATTEMPTED_FLAG  \
PLAYER_ID
201939           16-24 ft.            17     2   172                   1
201939             24+ ft.            26   116   239                   1
201939             24+ ft.            22   225    28                   1
201939             24+ ft.            23  -235     8                   1
201939           16-24 ft.            22   193   109                   1
…                      …           …    …    …                 …
201939             24+ ft.            26   175   195                   1
201939             24+ ft.            25  -163   195                   1
201939     Less Than 8 ft.             7    -3    75                   1
201939            8-16 ft.            13   -92   103                   1
```

```
201939            24+ ft.          28   168   236                   1

          SHOT_MADE_FLAG GAME_DATE  HTM  VTM
PLAYER_ID
201939              True  20190213  POR  GSW
201939             False  20190213  POR  GSW
201939             False  20190213  POR  GSW
201939             False  20190213  POR  GSW
201939             False  20190213  POR  GSW
…                    …         …   …   …
201939             False  20181223  GSW  LAC
201939             False  20181223  GSW  LAC
201939             False  20181223  GSW  LAC
201939              True  20181223  GSW  LAC
201939             False  20181223  GSW  LAC

[1340 rows x 23 columns]
```

[15]:
```python
grader.check("q1d")
```

[15]: q1d results: All test cases passed!

1.2 Question 2: Visualization

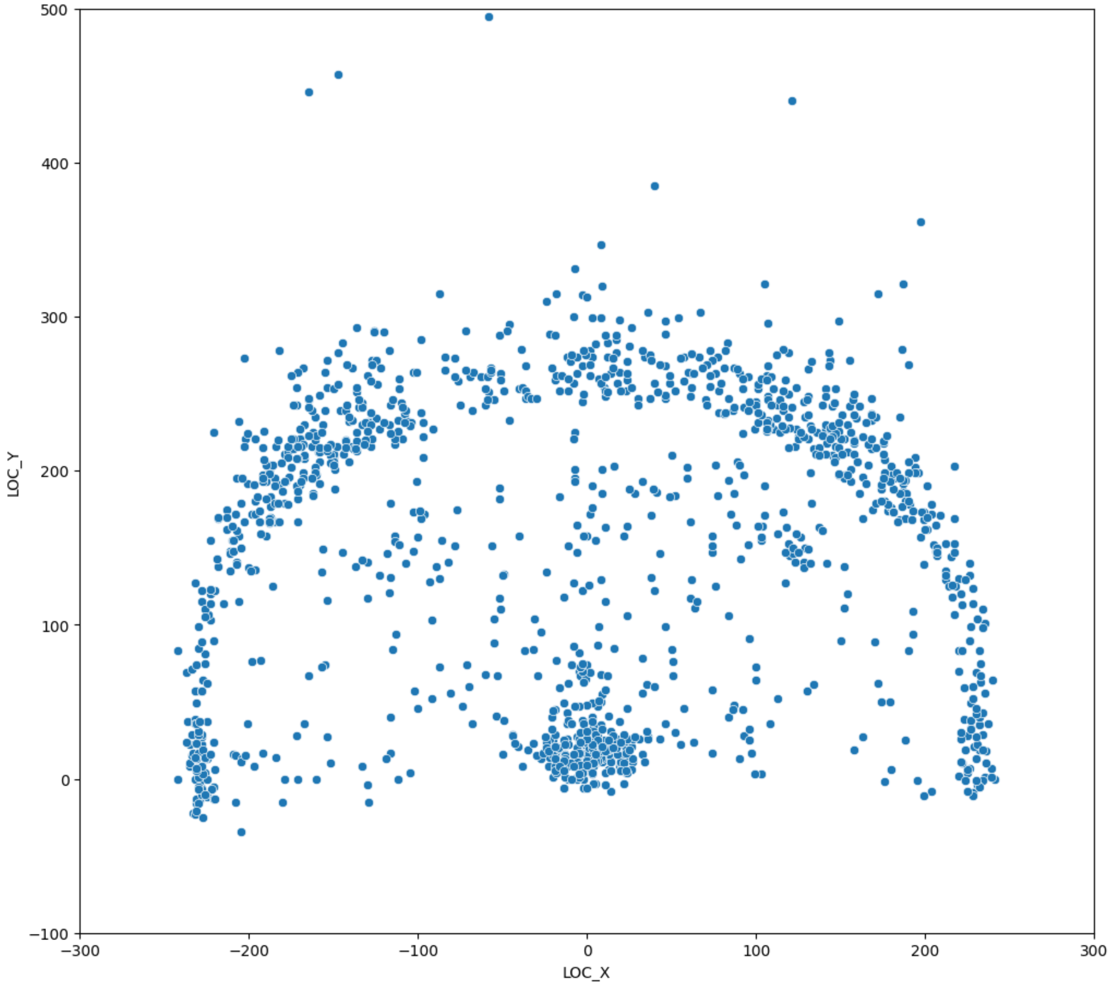
1.2.1 Question 2a: All Shots Scatter Plot

Use `seaborn` to create scatter plot of the location of Stephen Curry's shot attempts from this year (`LOC_X` and `LOC_Y`). When you call a scatterplot, seaborn returns a figure in an object, we'll call it `ax`. We can set properties of the figure by calling methods on `ax`. Use this approach to set the x-axis limits to span (-300, 300), the y-axis limits to span (-100, 500).

[16]:
```python
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=[12, 11])
ax2a = sns.scatterplot(x="LOC_X", y="LOC_Y", data=curry_data)
# Set x/y limits and labels
ax2a.set_xlim(-300, 300)
ax2a.set_ylim(-100,500)
plt.show()
```

```
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
```

```
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



[17]:
```python
grader.check("q2a")
```

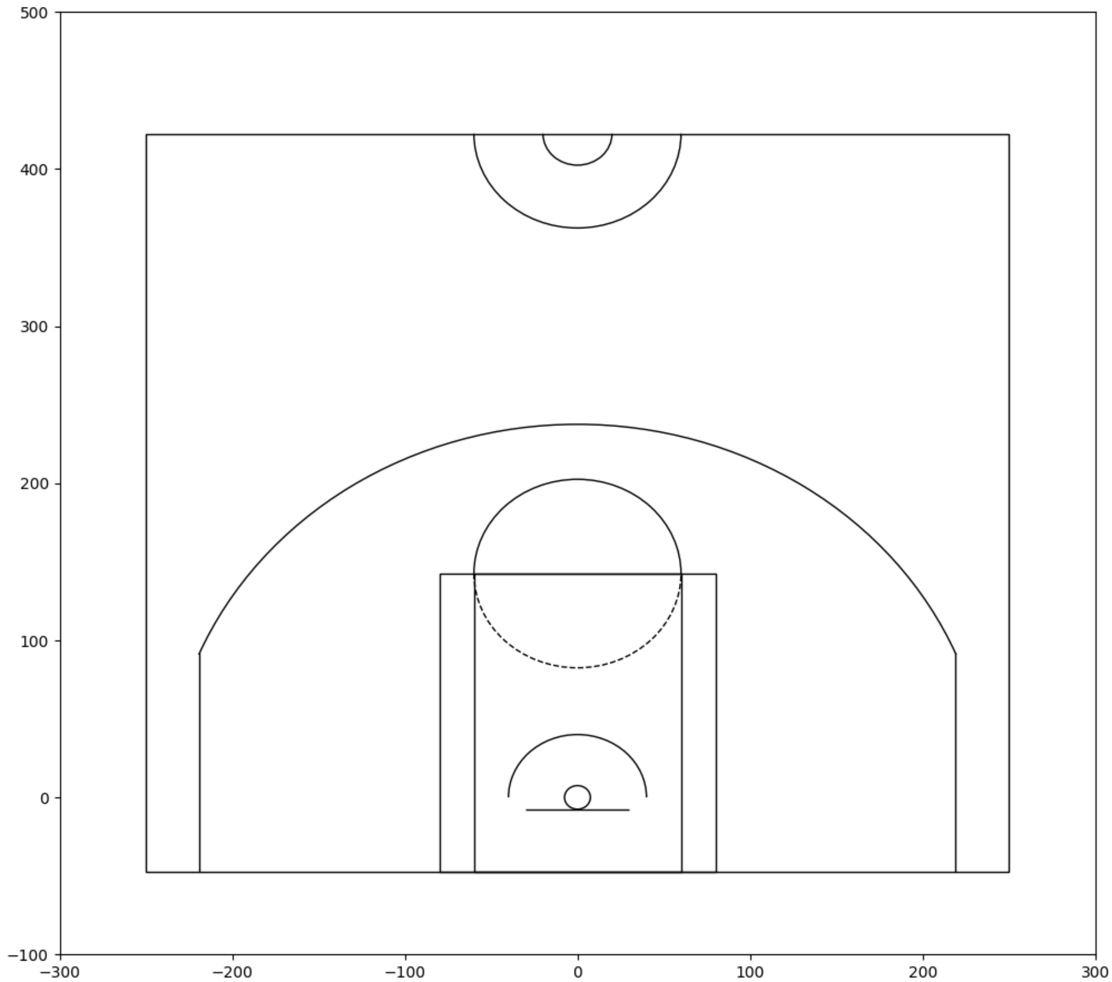
[17]: q2a results: All test cases passed!

Understanding any dataset is difficult without context. Lets add some important context by adding the relevant court lines into our diagram. If you are interested, you can read more about the lines and dimensions on the NBA basketball court. We will use code from `http://savvastjortjoglou.com/nba-shot-sharts.html` to add the court markings to our diagram. The `draw_court` function below will do this for us. The below cell will generate an example court.

[18]:
```python
## code is from http://savvastjortjoglou.com/nba-shot-sharts.html
def draw_court(ax=None, color='black', lw=1, outer_lines=False):

    from matplotlib.patches import Circle, Rectangle, Arc
```

```python
from matplotlib.pyplot import gca

# If an axes object isn't provided to plot onto, just get current one
if ax is None:
    ax = gca()

# Create the various parts of an NBA basketball court

# Create the basketball hoop
# Diameter of a hoop is 18" so it has a radius of 9", which is a value
# 7.5 in our coordinate system
hoop = Circle((0, 0), radius=7.5, linewidth=lw, color=color, fill=False)

# Create backboard
backboard = Rectangle((-30, -7.5), 60, 0, linewidth=lw, color=color)

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

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

# Three point line
# Create the side 3pt lines, they are 14ft long before they begin to arc
corner_three_a = Rectangle((-219, -47.5), 0, 140, linewidth=lw,
                           color=color)
corner_three_b = Rectangle((219, -47.5), 0, 140, linewidth=lw, color=color)
# 3pt arc - center of arc will be the hoop, arc is 23'9" away from hoop
# I just played around with the theta values until they lined up with the
# threes
three_arc = Arc((0, 0), 475, 475, theta1=22.5, theta2=157.5, linewidth=lw,
                color=color)

# Center Court
center_outer_arc = Arc((0, 422.5), 120, 120, theta1=180, theta2=0,
```

```
                           linewidth=lw, color=color)
    center_inner_arc = Arc((0, 422.5), 40, 40, theta1=180, theta2=0,
                           linewidth=lw, color=color)

    # List of the court elements to be plotted onto the axes
    court_elements = [hoop, backboard, outer_box, inner_box, top_free_throw,
                      bottom_free_throw, restricted, corner_three_a,
                      corner_three_b, three_arc, center_outer_arc,
                      center_inner_arc]

    if outer_lines:
        # Draw the half-court line, baseline and side-out bound lines
        outer_lines = Rectangle((-250, -47.5), 500, 470, linewidth=lw,
                                color=color, fill=False)
        court_elements.append(outer_lines)

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

    return ax

plt.figure(figsize=(12,11))
draw_court(outer_lines=True)
plt.xlim(-300,300)
plt.ylim(-100,500)
plt.show()
```

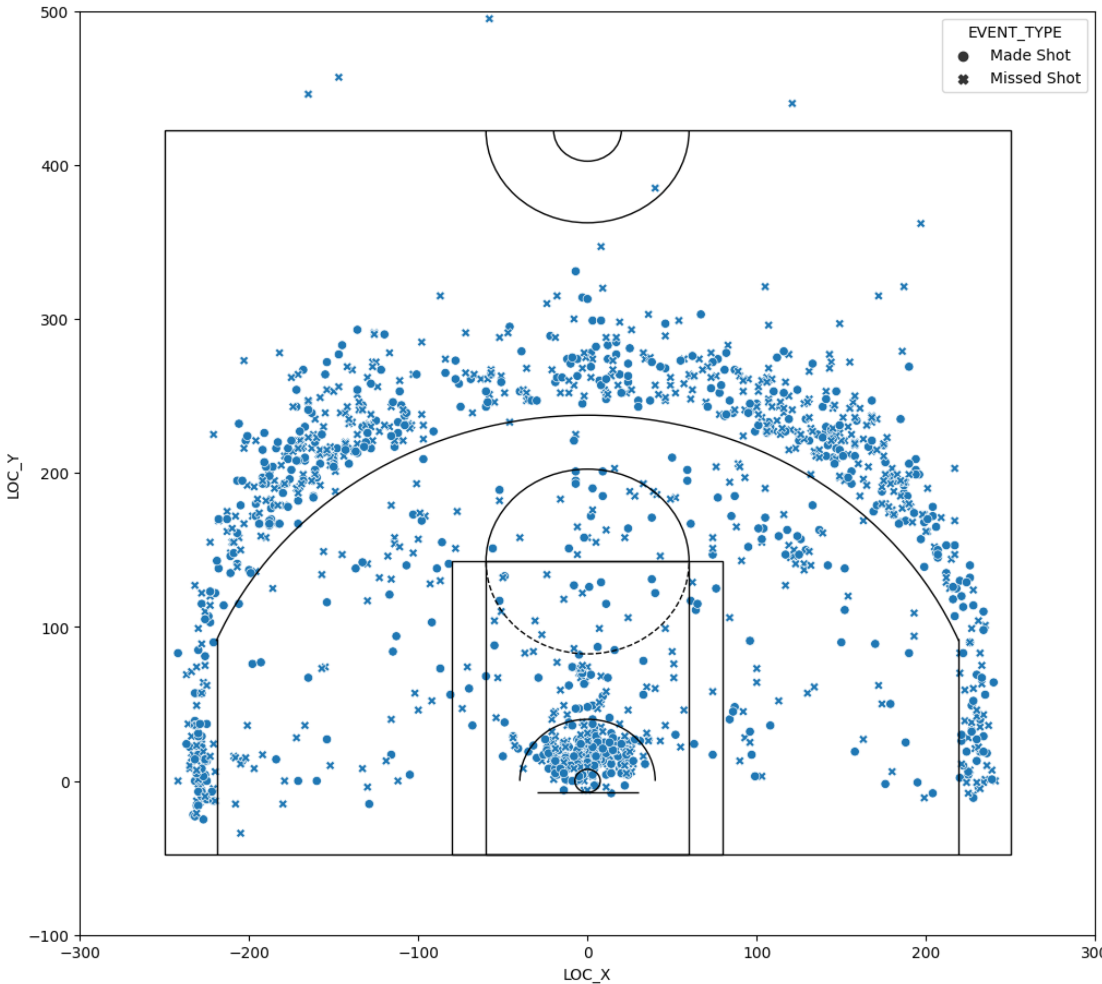


1.2.2 Question 2b: All Shots Scatter Plot + Court Outline

Again use seaborn to make a scatter plot of Stephen Curry's shots. Again, set the x-axis limits to span (-300, 300), the y-axis limits to span (-100, 500) color the points by whether the shot was made or missed. Set the missed shots to have an 'x' symbol and made shots to be a circular symbol. Call the `draw_court` function with `outer_lines` set to to be true. Save the `Axes` returned by the plot call in a variable called `ax`.

[19]:
```
plt.figure(figsize=(12, 11))
markers = {0 : "X", 1 : "o"}
ax = sns.scatterplot(x="LOC_X", y="LOC_Y", data=curry_data, style="EVENT_TYPE")

draw_court(outer_lines=True)
ax.set_xlim(-300, 300)
ax.set_ylim(-100,500)
plt.show()
```

```
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



1.2.3 Question 2c: Analyzing the Visualization

In a few sentences, discuss what makes this an effective or ineffective visualization for understanding the types of shots that Stephen Curry likes to take and is good at taking, relative to other players in the league. Are there ways it can be improved?

SOLUTION

This is a very effective visualiazation for understadning the types of shots that Curry likes to take and is good at taking becasue it shows us location n the vourt and whether he made it or not. A potential improvement to this chart is showing progression over time to see if he is consistent in the shots that he likes to take.

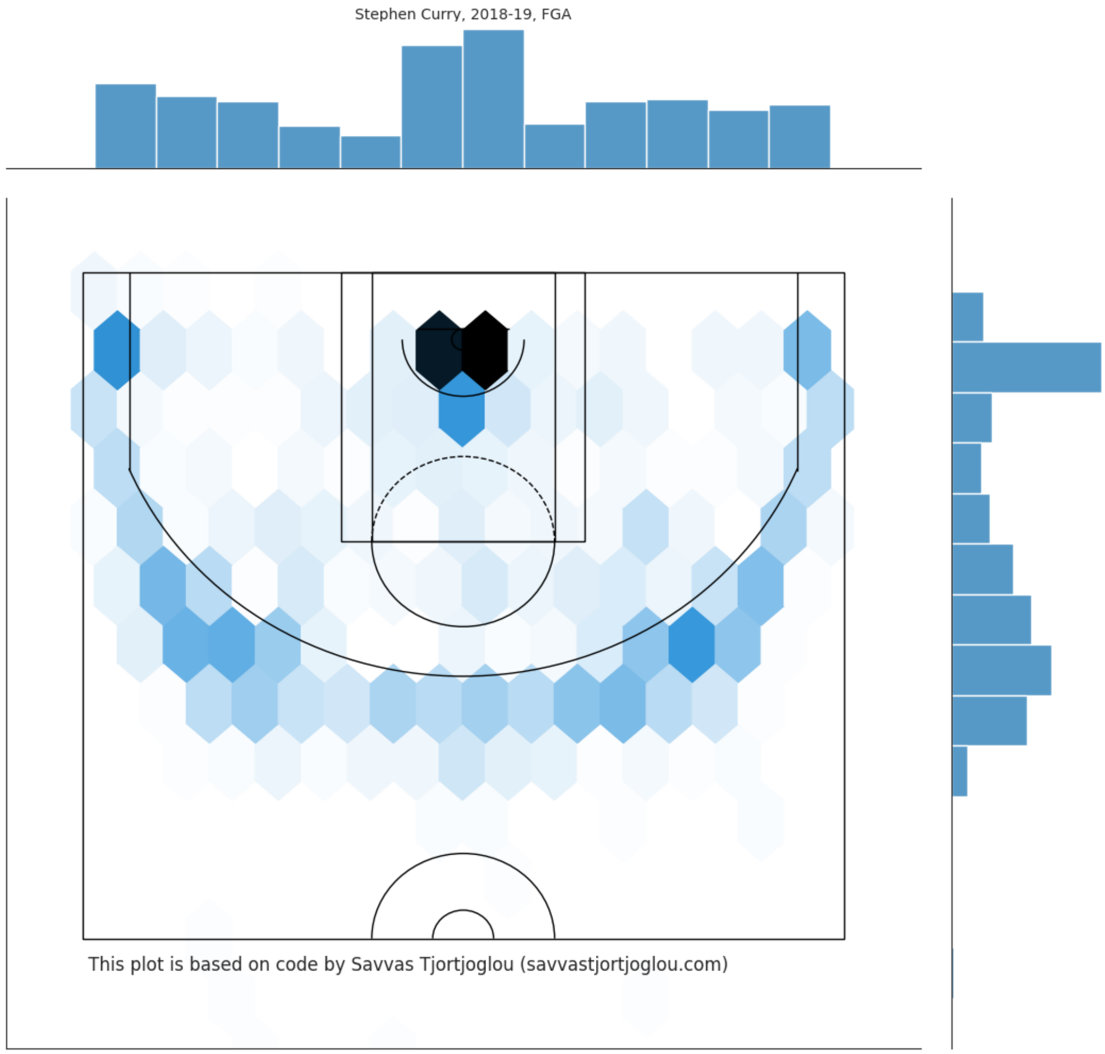
1.2.4 Question 2d: A Hexbin plot

Visualize Stephen Curry's shots by using a hexbin plot with marginal histograms. Also refer to setting figure aesthetics for what commands below do.

[20]:
```
sns.set_style("white")
joint_shot_chart = sns.jointplot(x="LOC_X", y="LOC_Y", data=curry_data, kind =␣
 ↪"hex")
joint_shot_chart.fig.set_size_inches(12,11)

# A joint plot has 3 Axes, the first one called ax_joint
# is the one we want to draw our court onto and adjust some other settings
ax = joint_shot_chart.ax_joint
draw_court(ax, outer_lines=True)

# Adjust the axis limits and orientation of the plot in order
# to plot half court, with the hoop by the top of the plot
ax.set_xlim(-300, 300)
ax.set_ylim(500, -100)

# Get rid of axis labels and tick marks
ax.set_xlabel('')
ax.set_ylabel('')
ax.tick_params(labelbottom=False, labelleft=False)

# Add a title
ax.set_title('Stephen Curry, 2018-19, FGA',
             y=1.2, fontsize=10)

# Add Data Source and Author
ax.text(-250,445,'\n This plot is based on code by Savvas Tjortjoglou␣
 ↪(savvastjortjoglou.com)',
        fontsize=12);
```

```
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
```

```
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



1.3 Question 3: Binning and Smoothing Shots

So far, in we have worked with dataframes which represent each shot as a single observation (row) within the dataset. However, this isn't a convenient data structure for the kinds of spatial analyses we will pursue below.

In this part, we will divide the court into square regions and create a matrix which includes the number of shots taken by a player in that region. We divide the court up into square bins (i.e. a 2d histogram) and, for each player, count number of shots that fall into each bin. Fortunately, this function is relatively simple to write using `numpy` module.

1.3.1 Question 3a: 2D Smoothing

Fill in the `bin_shots` function below. Use `np.histgram2d` to count count the shots in each bin. The bins are defined `bin_edges` which is a pandas Series of the form `(xedges, yedges)`. If `density = True`, call `ndimage.gaussian_filter` on the result of `np.histogram2d` with smoothing parameter `sigma`. This will create a smoothed version of the raw data histograms.

[21]:
```
def bin_shots(df, bin_edges, density=False, sigma=1):

    """Given data frame of shots, compute a 2d matrix of binned counts is␣
 ↪computed

    Args:
        df: data frame of shotchartdetail from nba.com.
            At the minimum, variables named LOCX and LOCY are required.
        bin_edges: bin edge definition: edges in x and edges in y

    Returns:
        binned: counts
        xedges: bin edges in X direction
        yedges: bin edges in Y direction
    """
    import numpy as np
    from scipy import ndimage

    ## Call np.histogram2d
    binned, xedges, yedges = np.histogram2d(x=df["LOC_X"], y=df["LOC_Y"],␣
 ↪bins=bin_edges)

    if density:

        # Recompute 'binned' using "gaussian_filter"
        binned = ndimage.gaussian_filter(binned, sigma)

        # Normalize the histogram to be a "density", e.g. mass across all bins␣
 ↪sums to 1.
        binned /= np.sum(binned)

    return(binned, xedges, yedges)
```

[22]:
```
grader.check("q3a")
```

[22]:
```
q3a results: All test cases passed!
```

1.3.2 Question 3b: Visualize the binning on curry_data

Call `bin_shots` on `curry_data` to create a binned but unsmoothed matrix of shot counts (call this `curry_binned_unsmoothed`), a binned and smoothed matrix of counts with `sigma=1` (call this `curry_binned_smoothed1`) and one with `sigma=5` (call this `curry_binned_smoothed5`). Use the bin edges defined below:

[23]:
```
## bin edge definitions in inches
xedges = np.linspace(start=-300, stop=300, num=151)
yedges = np.linspace(start=-48, stop=372, num=106)
```

Type your answer here, replacing this text.

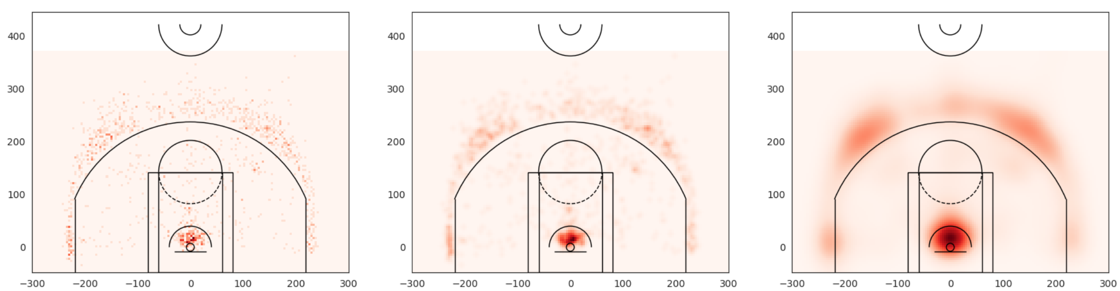
[24]:
```
bin_edges = (xedges, yedges)

curry_binned_unsmoothed, xe, ye = bin_shots(curry_data, bin_edges)
curry_binned_smoothed1,  xe, ye = bin_shots(curry_data, bin_edges, density=True)
curry_binned_smoothed5,  xe, ye = bin_shots(curry_data, bin_edges,␣
 ↪density=True, sigma=5)
print(curry_binned_smoothed5)
```

```
[[1.95344898e-07 2.12205372e-07 2.45323183e-07 … 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [2.81487940e-07 3.05605557e-07 3.52944988e-07 … 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [4.76489083e-07 5.17025451e-07 5.96525158e-07 … 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 …
 [1.76131713e-07 2.07432432e-07 2.70768188e-07 … 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [1.08294620e-07 1.27698120e-07 1.66978121e-07 … 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [7.77901305e-08 9.18314872e-08 1.20268962e-07 … 0.00000000e+00
  0.00000000e+00 0.00000000e+00]]
```

[25]:
```
def plot_shotchart(binned_counts, xedges, yedges, ax=None, use_log=False, cmap␣
 ↪= 'Reds'):

    """Plots 2d heatmap from vectorized heatmap counts

    Args:
        hist_counts: vectorized output of numpy.histogram2d
        xedges, yedges: bin edges in arrays
        ax: figure axes [None]
        use_log: will convert count x to log(x+1) to increase visibility [False]
        cmap: Set the color map https://matplotlib.org/examples/color/
 ↪colormaps_reference.html
    Returns:
```

```
        ax: axes with plot
    """

    import numpy as np
    import matplotlib.pyplot as plt

    ## number of x and y bins.
    nx = xedges.size - 1
    ny = yedges.size - 1

    X, Y = np.meshgrid(xedges, yedges)

    if use_log:
        counts = np.log(binned_counts + 1)

    if ax is None:
        fig, ax = plt.subplots(1,1)

    ax.pcolormesh(X, Y, binned_counts.T, cmap=cmap)
    ax.set_aspect('equal')

    draw_court(ax)

    return(ax)
```

[26]:
```
fig, ax = plt.subplots(1, 3, figsize=(20,60))

plot_shotchart(curry_binned_unsmoothed, xedges, yedges, ax=ax[0])
plot_shotchart(curry_binned_smoothed1, xedges, yedges, ax=ax[1])
plot_shotchart(curry_binned_smoothed5, xedges, yedges, ax=ax[2])
fig.show()
```



1.3.3 Vectorize Shot Images

- Here we proceed create a dictionary of smoothed patterns, each vectorized into a 1-d array (like Lab 6)

- In this case, the object `all_smooth` is a dictionary that consists of arrays of length `15750`.
- Each entry in `all_smooth` represents the smoothed frequency of shots along the bins generated in the code above for a given player.

[27]:
```
## number of bins is one less than number of edges (remember homework 1)
nx = xedges.size - 1
ny = yedges.size - 1

## 2d histogram containers for binned counts and smoothed binned counts
all_counts = []
all_smooth = []
pids = []

## 2d histogram containers for binned counts and smoothed binned counts

## data matrix: players (row) by vectorized 2-d court locations (column)
for i, one in enumerate(allshots.groupby('PLAYER_ID')):

    ## what does this line do?
    pid, pdf = one

    num_shots = len(pdf.index)
    if(num_shots > 100):

        tmp1, xedges, yedges = bin_shots(pdf, bin_edges=(xedges, yedges),␣
↪density=True, sigma=2)
        tmp2, xedges, yedges = bin_shots(pdf, bin_edges=(xedges, yedges),␣
↪density=False)

        ## vectorize and store into list
        all_smooth += [tmp1.reshape(-1)]
        all_counts += [tmp2.reshape(-1)]
        pids += [pid]

X = np.vstack(all_smooth).T
p, n = X.shape

print('Number of shot regions (p):', p)
print('Number of players (n):', n)
```

```
Number of shot regions (p): 15750
Number of players (n): 388
```

1.4 Question 4: Non-negative Matrix Factorization (NMF)

The non-negative matrix factorization is a dimension reduction technique that is often applied to image data. It is similar to PCA except that is only applicable for strictly positive data. We can apply the NMF to vectorized versions of the shot surface. This is useful because we can convert the observed matrix of shot surfaces into: * Bases: Identifying modes of shooting style (number of modes is determined by `n_components` argument to `NMF` function below) * Coefficients: How each players shooting style could be expressed as a (positive) linear combination of these bases

The NMF solves the following problem: given some matrix X is $p \times n$ matrix, NMF computes the following factorization:

$$\min_{W,H} \|X - WH\|_F \text{ subject to } W \geq 0,\ H \geq 0,$$

where W is $p \times r$ matrix and H is $r \times n$ matrix.

In this homework, we have the following:

The data matrix X X is of dimension n={number of players} and p={number of total square bins on the court}. Each column corresponds to a player, with entries corresponding to a "flattened" or "vectorized" version of the 2d histograms plotted in part 4b.

Bases matrix: W Columns W_i contain the shot "bases". First, we will try it with $r = 3$ bins in 5a, and then with $r = 10$ bins in 5d.

Coefficient matrix: H Each column of H gives a coefficient for each of the bases vectors in W, and there are n columns for each player.

The `sklearn` library is one of the main Python machine learning libraries. It has a built in NMF function for us. The function below runs this function and normalizes the basis surfaces to sum to 1.

[28]:
```
## Non-negative Matrix Factorization
def non_negative_matrix_decomp(n_components, array_data):
    import sklearn.decomposition as skld
    model = skld.NMF(n_components=n_components, init='nndsvda', max_iter=500,␣
 ↪random_state=0)
    W = model.fit_transform(array_data)

    # Normalize basis vectors to sum to 1
    Wsum = W.sum(axis=0)
    W = W/Wsum

    ## fix H correspondingly
    H = model.components_
    H = (H.T * Wsum).T

    nmf = (W, H)
    return(nmf)
```

1.4.1 Question 4a: Computing NMF Factorization

Compute the NMF on all player's shot charts, X, assuming with `n_components` = 3 (i.e. each shot chart can be represented as a positive linear combination of 3 "basis" shot charts). Fill in `plot_vectorized_shot_chart`. This takes a the a vector of binned shot counts, converts it back to a matrix of the appropriate size and then calls `plot_shotchart` on the matrix. The numpy function `reshape` will be useful here: https://docs.scipy.org/doc/numpy/reference/generated/numpy.reshape.html

[29]:
```
W3, H3 = non_negative_matrix_decomp(3, X)
```

[30]:
```
grader.check("q4a")
```

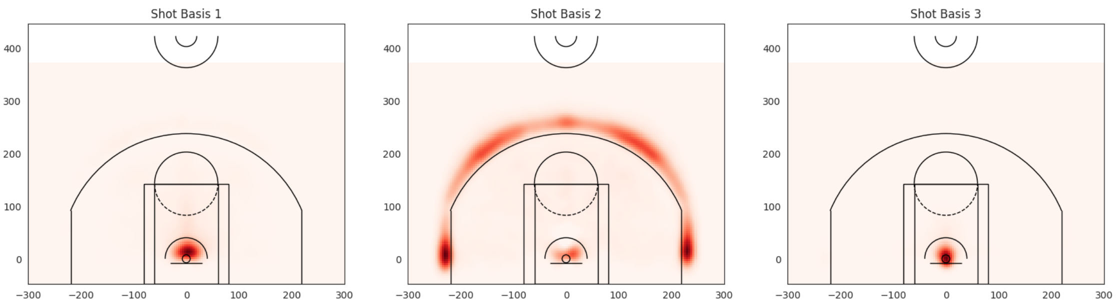
[30]:
```
q4a results: All test cases passed!
```

1.4.2 Question 4b: Visualizing Shot Types

Plot the first three basis images by calling `plot_vectorized_shot_chart` below on the columns of `W3`.

[31]:
```
def plot_vectorized_shotchart(vec_counts, xedges, yedges, ax=None,␣
 ↪use_log=False, cmap = 'Reds'):

    """Plots 2d heatmap from vectorized heatmap counts

    Args:
        hist_counts: vectorized output of numpy.histogram2d
        xedges, yedges: bin edges in arrays
        ax: figure axes [None]
        use_log: will convert count x to log(x+1) to increase visibility [False]
        cmap: Set the color map https://matplotlib.org/examples/color/
 ↪colormaps_reference.html
    Returns:
        ax: axes with plot
    """

    nx = xedges.size - 1
    ny = yedges.size - 1

    # use reshape to convert a vectorized counts back into a 2d histogram
    two_d_counts = vec_counts.reshape(nx,ny)

    return(plot_shotchart(two_d_counts, xedges, yedges, ax=ax, use_log=use_log,␣
 ↪cmap=cmap))

fig, ax = plt.subplots(1, 3, figsize=(20,60))

## Write a for loop
for i in range(3):
    nx, ny = xedges.size-1, yedges.size-1
    two_d_counts = W3[:,i].reshape(nx,ny)
    # Call plot_vectorized_shot_chart
```

```
    plot_vectorized_shotchart(W3[:,i], xedges, yedges, ax=ax[i])
    ax[i].set_title('Shot Basis %i' % (i+1))
```



1.4.3 Question 4c: Reconstruction Error

Below we re-construct the shooting pattern for a single player. By "reconstructing" we mean use the approximation

$$\hat{X} = WH$$

obtained via NMF. Find $\hat{X}$ by multipling W and H. In python the @ symbol is used for matrix multiplication.

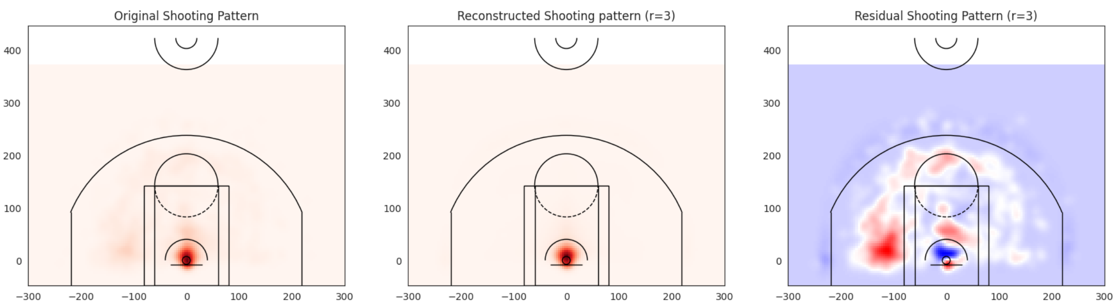
Type your answer here, replacing this text.

[32]:
```
X3_hat = W3@H3
print(X3_hat)
```

```
[[0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 …
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]]
```

[33]:
```
# Find the player_id of LaMarcus Aldridge
player_id = allplayers.query('DISPLAY_FIRST_LAST == "LaMarcus Aldridge"').index.
 ↪values[0]

## find index in X corresponding to that player
to_plot_idx = np.where(pids == player_id)[0][0]

fig, ax = plt.subplots(1, 3, figsize=(20,60))

## Call plot_vectorized_shot_chart
original_shotchart = plot_vectorized_shotchart(X[:, to_plot_idx], xedges,␣
 ↪yedges, ax=ax[0])
```

```
reconstructed_shotchart = plot_vectorized_shotchart(X3_hat[:, to_plot_idx],␣
 ↪xedges, yedges, ax=ax[1])
residual_chart = plot_vectorized_shotchart(X[:, to_plot_idx] - X3_hat[:,␣
 ↪to_plot_idx], xedges, yedges, ax=ax[2], cmap="bwr")

# print(max(abs(X3_hat[:, to_plot_idx] - X[:, to_plot_idx])))
ax[0].set_title('Original Shooting Pattern')
ax[1].set_title('Reconstructed Shooting pattern (r=3)')
ax[2].set_title('Residual Shooting Pattern (r=3)')
None # prevents the title string from showing up as output
```



1.4.4 Question 4d: Choice of Colormap

Why does it make sense to use a *sequential* palette for the original and reconstructed shot charts and a *diverging* palette for the residual? *Hint:* Read the introduction to colormaps here.

SOLUTION

Sequential palettes are useful when visualizing data with a set ordering. For our original and reconstructed shooting patterns, the instensity saturation values do a good job of representing the vectorized shot frequency. That is because a player cannot have less than zero shots at any given position, and they do not have an upper limit to how many shots they can make at any given position. A diverging color map makes sense for our residual plot since we are vizualizing the differences between the origin and reconstructed shooting patterns. Our data is centered around the value 0 which would mean that there is no difference in the shooting pattern. The diverging color map will allow us to use different colors to spot the differences between the two shooting patterns.

What areas of the court does this player to shoot more and where less relative to the reconstructed area. If its helpful, you can refer to court locations by name using this legend here.

SOLUTION

This player tends to shoot more in the short corners and the top of the key compared to the reconstructed area. The reconstructed shooting pattern shows that the player shoot more in the restricted area than the original shooting pattern.

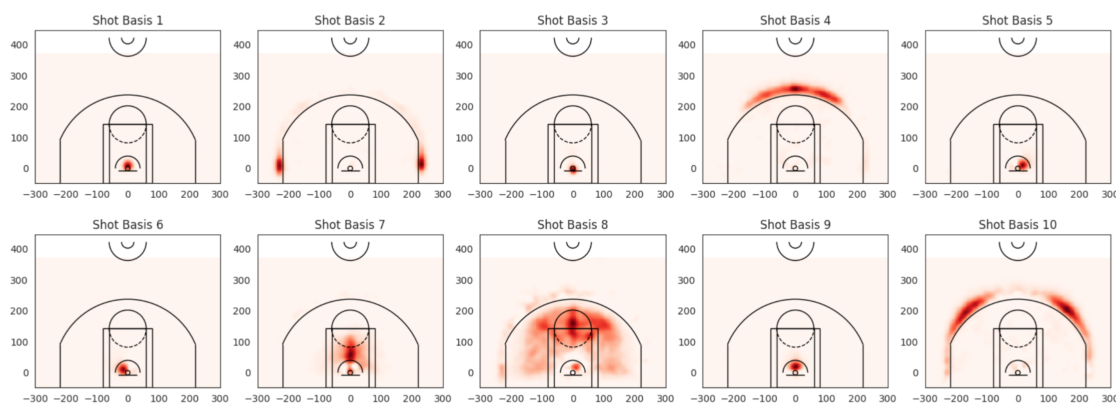
1.4.5 Question 4e: More Detailed Modeling

Re-run the analysis, this time for 10 basis vectors instead of 3. Again plot the bases using `plot_vectorized_shotchart` on the columns of `W10`.

Hint: Study the following code

```
fig, ax = plt.subplots(2, 5, figsize=(20, 7))
ax = ax.flatten() # turn ax into a flat array
ax[0].set_title('hello')
ax[9].set_title('there')
fig.show()
```

[34]:
```
W10, H10 = non_negative_matrix_decomp(10, X)

fig, ax = plt.subplots(2, 5, figsize=(20, 7))

## Write a for loop
for i in range(10):
    plot_vectorized_shotchart(W10[:,i], xedges, yedges, ax=ax[i//5, i % 5])
    ax[i//5, i % 5].set_title('Shot Basis %i' % (i+1))
```



If you did things correctly, you should be really impressed! We've identified potentially interesting patterns of shooting styles without actually specifying anything about the way basketball is played or where the relevant lines are on the court. The resulting images are based only on the actual behavior of the players. Even more impressive is that we're capturing similarity in regions that are far apart on the court. One reason we can do this is that a basketball court is symmetric along the length of the court (i.e. symmetric about x=0). However, people tend to be left or right hand dominant, which might affect their preferences. Look carefuly at the shot basis plots above: is there any evidence of *asymmetry* in player shooting behavior? Refer to specific basis images in your answer.

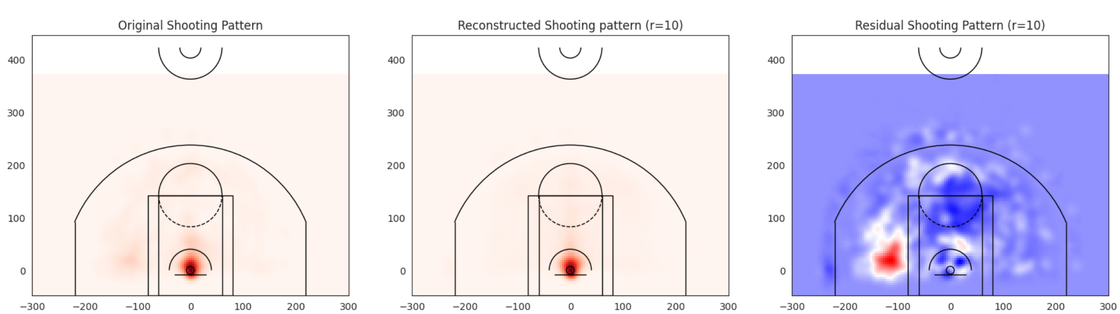
SOLUTION

In Shot Basis 8, it seems that players prefer to shoot 3-pointers from the left wing of the arc compared to the right. This is also demonstrated in Shot Basis 10 where there is a bigger spread of

3 point shots on the left side. This could be due to a majority of players being right-handed which means that that wing would be more comfortable for them to shoot 3-pointers.

Repeat part 5b, and again plot original, reconstructed and residual shot chats for LaMarcus Aldridge.

[35]:
```
X10_hat = W10@H10

fig, ax = plt.subplots(1, 3, figsize=(20,60))

# I took the first player appearing in first column
# (you probably want to do more interesting players)
## find index in X corresponding to that player
to_plot_idx = np.where(pids == player_id)[0][0]

original_shotchart = plot_vectorized_shotchart(X[:, to_plot_idx], xedges,␣
↪yedges, ax=ax[0])
reconstructed_shotchart = plot_vectorized_shotchart(X10_hat[:, to_plot_idx],␣
↪xedges, yedges, ax=ax[1])
residual_chart = plot_vectorized_shotchart(X[:, to_plot_idx] - X10_hat[:,␣
↪to_plot_idx], xedges, yedges, ax=ax[2], cmap="bwr")

ax[0].set_title('Original Shooting Pattern')
ax[1].set_title('Reconstructed Shooting pattern (r=10)')
ax[2].set_title('Residual Shooting Pattern (r=10)');
```



1.4.6 Question 4f: Comparing Players

With `H10` matrix, it is possible to compare any pair of players. For all players pairwise, i and j, compare using euclidean distance between their coefficients:

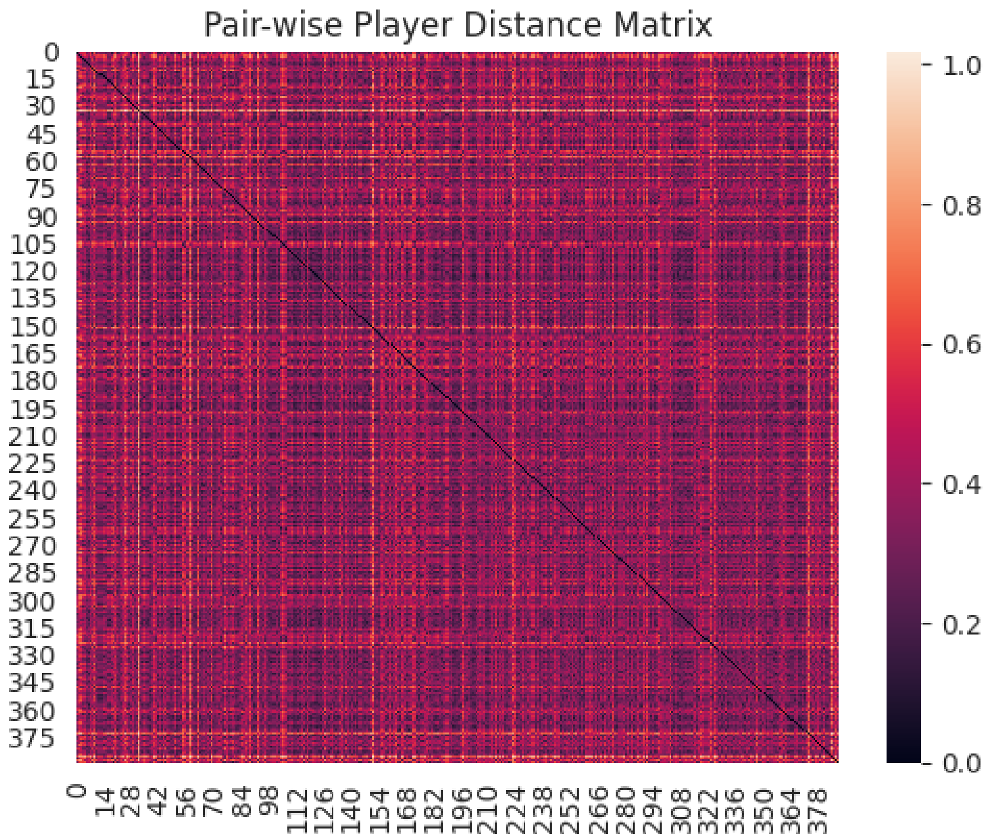
$$\text{player-distance}(i, j) = \|H_i - H_j\|_2 = \left(\sum_{k=1}^{10}(H_{ki} - H_{kj})^2\right)^{1/2}$$

Create a heatmap for comparing pair-wise player distance matrix. Find the two pairs of players with the smallest distances. Also, find two pairs of players with largest distances.

Hint: you can construct the distance matrix manually or use `scipy.spatial.distance_matrix`.

SOLUTION

[36]:
```
from scipy.spatial import distance_matrix

player_dists = distance_matrix(H10.T, H10.T)

sns.heatmap(player_dists)
plt.title("Pair-wise Player Distance Matrix")
plt.show()

rows=[]
for i in range(len(player_dists)):
    for j in range(len(player_dists)):
        if i!=j:
            rows.append([(i,j),player_dists[i,j]])
rows = pd.DataFrame(rows)
rows.columns=["players", "dist"]
smallest_ids = rows.nsmallest(4, columns="dist")
min_player1 = allplayers.loc[pids[smallest_ids.
 ↪iloc[0,0][0]]]["DISPLAY_FIRST_LAST"]
min_player2 = allplayers.loc[pids[smallest_ids.
 ↪iloc[0,0][1]]]["DISPLAY_FIRST_LAST"]
print("First smallest distance = " + min_player1 + ", " + min_player2)
min_player3 = allplayers.loc[pids[smallest_ids.
 ↪iloc[2,0][0]]]["DISPLAY_FIRST_LAST"]
min_player4 = allplayers.loc[pids[smallest_ids.
 ↪iloc[2,0][1]]]["DISPLAY_FIRST_LAST"]
print("Second smallest distance = " + min_player3 + ", " + min_player4)

largest_ids = rows.nlargest(4, columns="dist")
max_player1 = allplayers.loc[pids[largest_ids.
 ↪iloc[0,0][0]]]["DISPLAY_FIRST_LAST"]
max_player2 = allplayers.loc[pids[largest_ids.
 ↪iloc[0,0][1]]]["DISPLAY_FIRST_LAST"]
print("First largest distance = " + max_player1 + ", " + max_player2)
max_player3 = allplayers.loc[pids[largest_ids.
 ↪iloc[2,0][0]]]["DISPLAY_FIRST_LAST"]
max_player4 = allplayers.loc[pids[largest_ids.
 ↪iloc[2,0][1]]]["DISPLAY_FIRST_LAST"]
print("Second largest distance = " + max_player3 + ", " + max_player4)
```




```
First smallest distance = CJ McCollum, Jamal Murray
Second smallest distance = Malik Beasley, Malik Monk
First largest distance = PJ Tucker, Kyle O'Quinn
Second largest distance = Jose Calderon, PJ Tucker
```

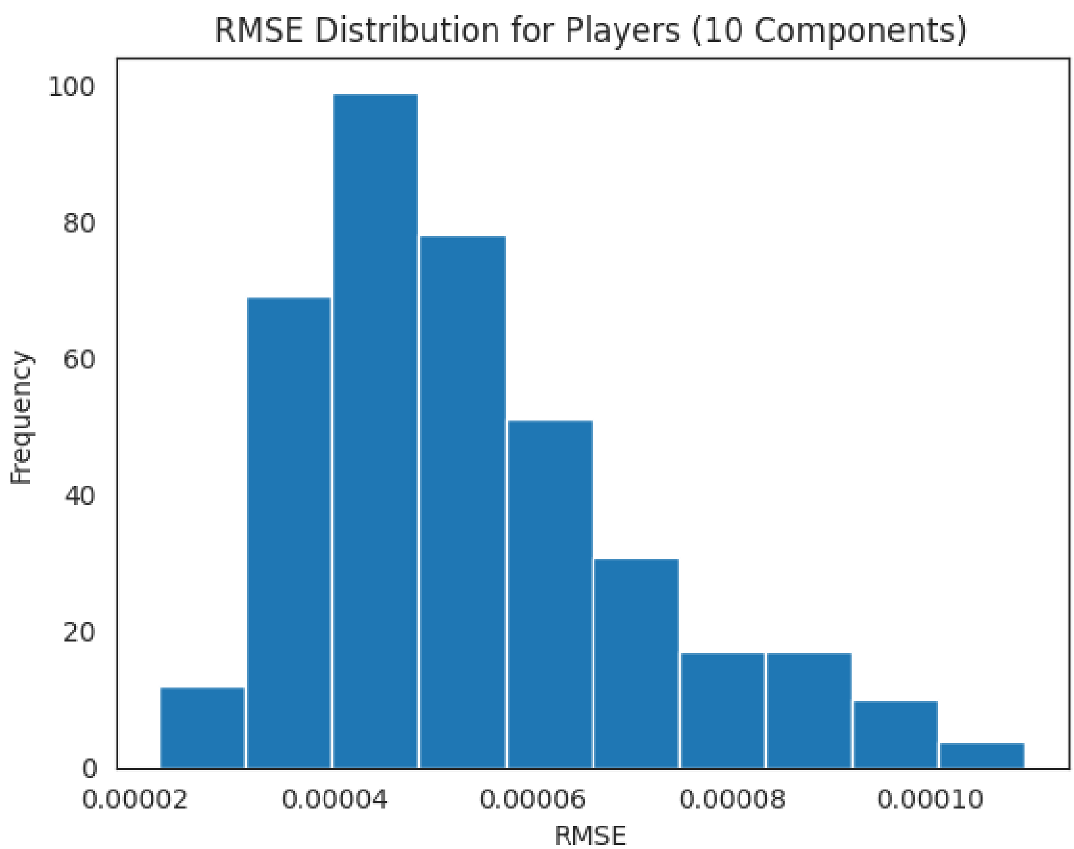
1.4.7 Question 4g: Residuals

The residual betwene `Xhat` and `X` gives a sense of how well a player is decribed by NMF computed matrices `W` and `H`. Calculate RMSE for each player, and plot the histogram. Comment on this distribution and players with smallest and largest RMSEs (use 10 components).

SOLUTION

[37]:
```
from sklearn.metrics import mean_squared_error
rmse_values = []
for i in range(n):
    rmse = np.sqrt(mean_squared_error(X[:, i], X10_hat[:, i]))
    rmse_values.append(rmse)
plt.hist(rmse_values)
plt.title('RMSE Distribution for Players (10 Components)')
plt.xlabel('RMSE')
```

```
plt.ylabel('Frequency')
plt.show()

min_RMSE, max_RMSE = np.argmin(rmse_values), np.argmax(rmse_values)
min_RMSE_index, max_RMSE_index = pids[min_RMSE], pids[max_RMSE]
min_RMSE_player = allshots.query('PLAYER_ID == ' +␣
 ↪str(min_RMSE_index))["PLAYER_NAME"].iloc[0]
max_RMSE_player = allshots.query('PLAYER_ID == ' +␣
 ↪str(max_RMSE_index))["PLAYER_NAME"].iloc[0]
print("The player with the lowest RMSE is " + min_RMSE_player)
print("The player with the highest RMSE is " + max_RMSE_player)
```




```
The player with the lowest RMSE is Montrezl Harrell
The player with the highest RMSE is Brad Wanamaker
```

The overall RMSE peaks around 0.00004 for most players which makes our histogram right skewed. The highest observed RMSE is around 0.00010 and the lowest RMSE is around 0.00002. So our Xhat does a very good job of approximating the actual shot patterns.

1.4.8 Question 4h: Proposing improvements

One of the main purposes of exploratory data analysis is to generate new ideas, directions, and hypothesis for future analyses and experiments. Take two players of your choice and compare their shooting patterns with various visualizations.

State any insights and defend your conclusions with visual and/or numerical comparisons.

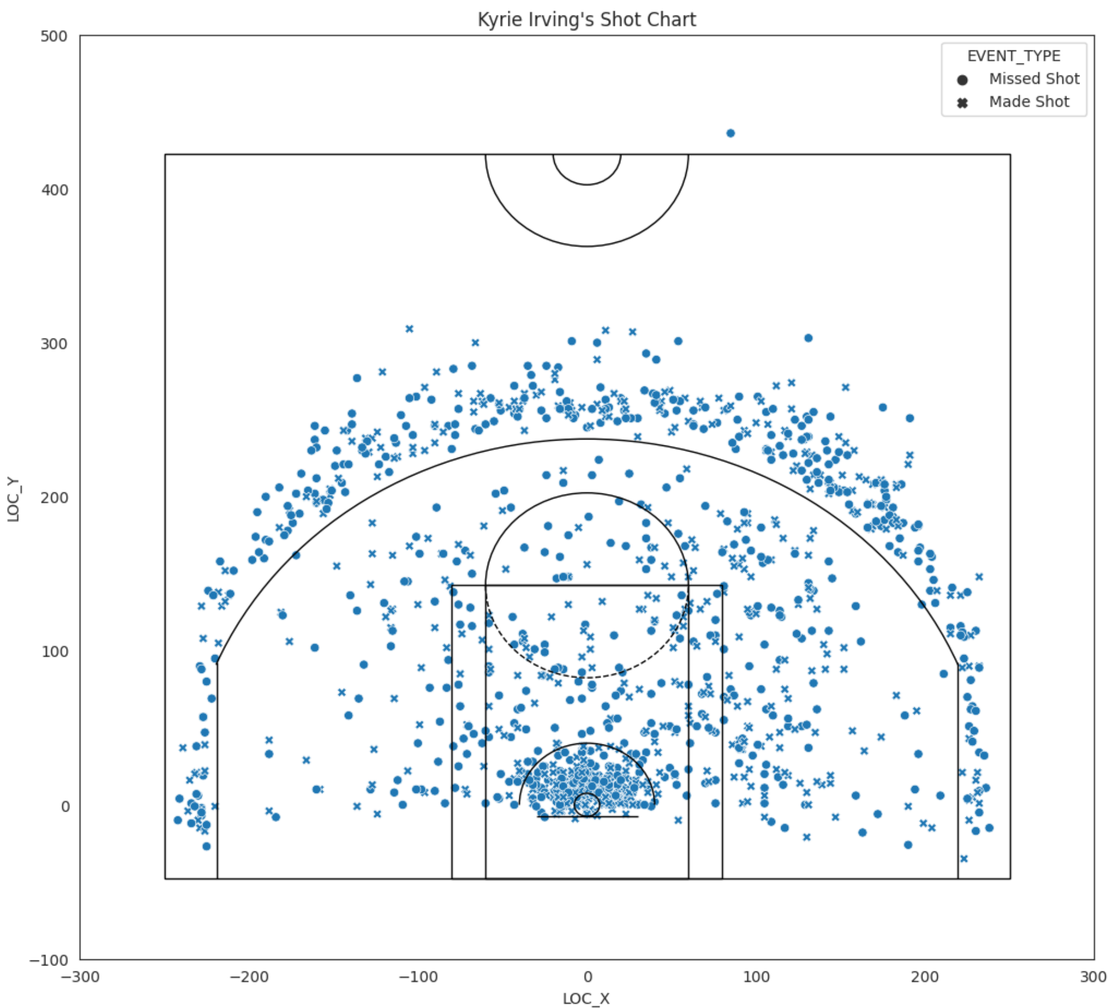
SOLUTION

[38]:
```
query_str = 'DISPLAY_FIRST_LAST == "Kyrie Irving"'
irving_id = str(allplayers.query(query_str).index.values[0])
irving_data = allshots.query('PLAYER_ID == ' + irving_id).
↪astype({"SHOT_MADE_FLAG" : "bool"})

query_str = 'DISPLAY_FIRST_LAST == "LeBron James"'
james_id = str(allplayers.query(query_str).index.values[0])
james_data = allshots.query('PLAYER_ID == ' + james_id).
↪astype({"SHOT_MADE_FLAG" : "bool"})

# KYRIE IRVING
plt.figure(figsize=(12, 11))
axI = sns.scatterplot(x="LOC_X", y="LOC_Y", data=irving_data,␣
↪style="EVENT_TYPE")
draw_court(outer_lines=True)
axI.set_xlim(-300, 300)
axI.set_ylim(-100,500)
plt.title("Kyrie Irving's Shot Chart")
plt.show()

sns.set_style("white")
joint_shot_chart = sns.jointplot(x="LOC_X", y="LOC_Y", data=irving_data, kind =␣
↪"hex")
joint_shot_chart.fig.set_size_inches(12,11)

axI2 = joint_shot_chart.ax_joint
draw_court(axI2, outer_lines=True)

axI2.set_xlim(-300, 300)
axI2.set_ylim(500, -100)

# Get rid of axis labels and tick marks
axI2.set_xlabel('')
axI2.set_ylabel('')
axI2.tick_params(labelbottom=False, labelleft=False)

# Add a title
axI2.set_title('Kyrie Irving, 2018-19, FGA',
            y=1.2, fontsize=10)
```

```
# Add Data Source and Author
axI2.text(-250,445,'\n This plot is based on code by Savvas Tjortjoglou␣
↪(savvastjortjoglou.com)',
        fontsize=12);


# KLAY THOMPSON
plt.figure(figsize=(12, 11))
axJ = sns.scatterplot(x="LOC_X", y="LOC_Y", data=james_data, style="EVENT_TYPE")
draw_court(outer_lines=True)
axJ.set_xlim(-300, 300)
axJ.set_ylim(-100,500)
plt.title("LeBron James' Shot Chart")
plt.show()

sns.set_style("white")
joint_shot_chart = sns.jointplot(x="LOC_X", y="LOC_Y", data=james_data, kind =␣
↪"hex")
joint_shot_chart.fig.set_size_inches(12,11)

axJ2 = joint_shot_chart.ax_joint
draw_court(axJ2, outer_lines=True)

axJ2.set_xlim(-300, 300)
axJ2.set_ylim(500, -100)

# Get rid of axis labels and tick marks
axJ2.set_xlabel('')
axJ2.set_ylabel('')
axJ2.tick_params(labelbottom=False, labelleft=False)

# Add a title
axJ2.set_title('LeBron James, 2018-19, FGA',
             y=1.2, fontsize=10)

# Add Data Source and Author
axJ2.text(-250,445,'\n This plot is based on code by Savvas Tjortjoglou␣
↪(savvastjortjoglou.com)',
        fontsize=12);
```

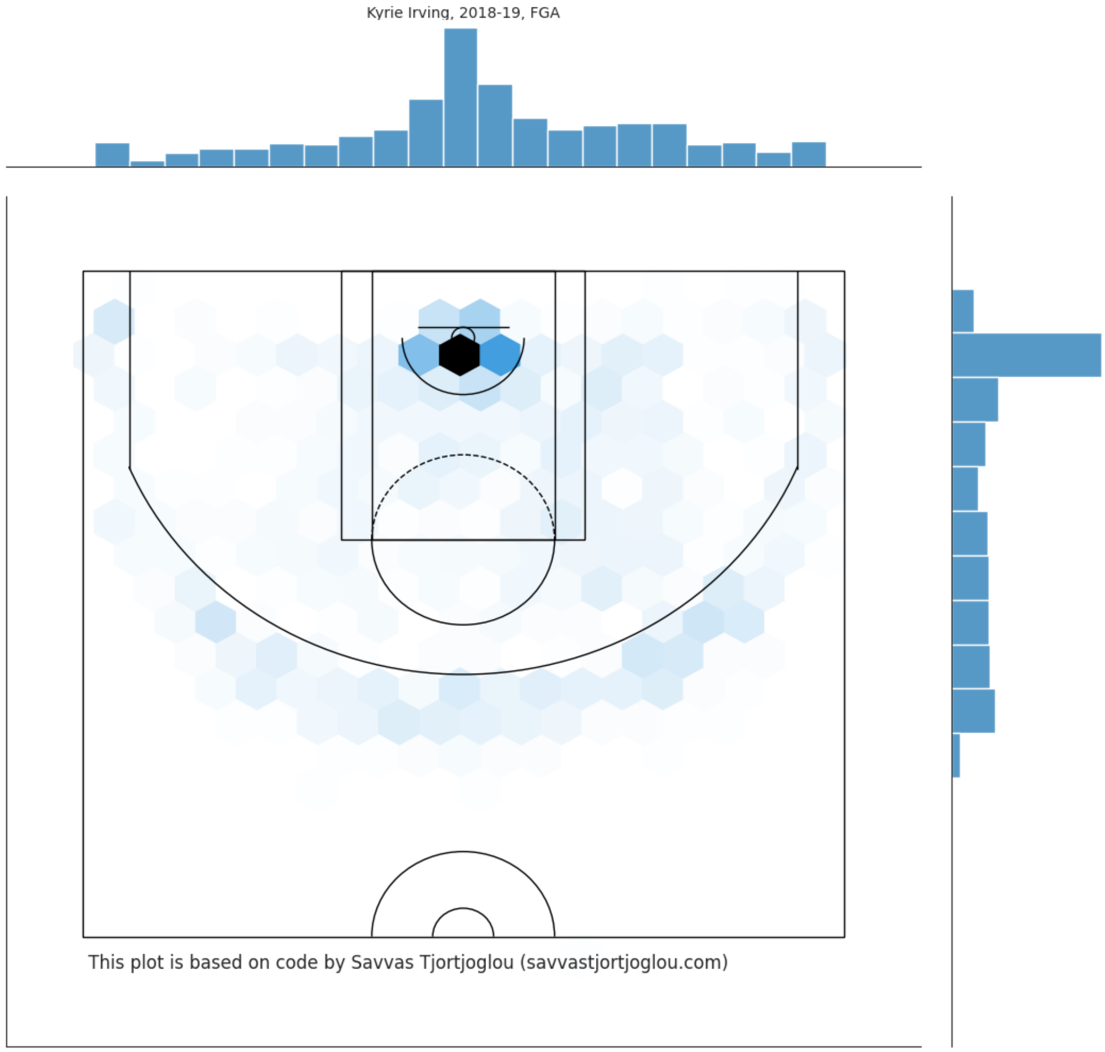
```
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
```

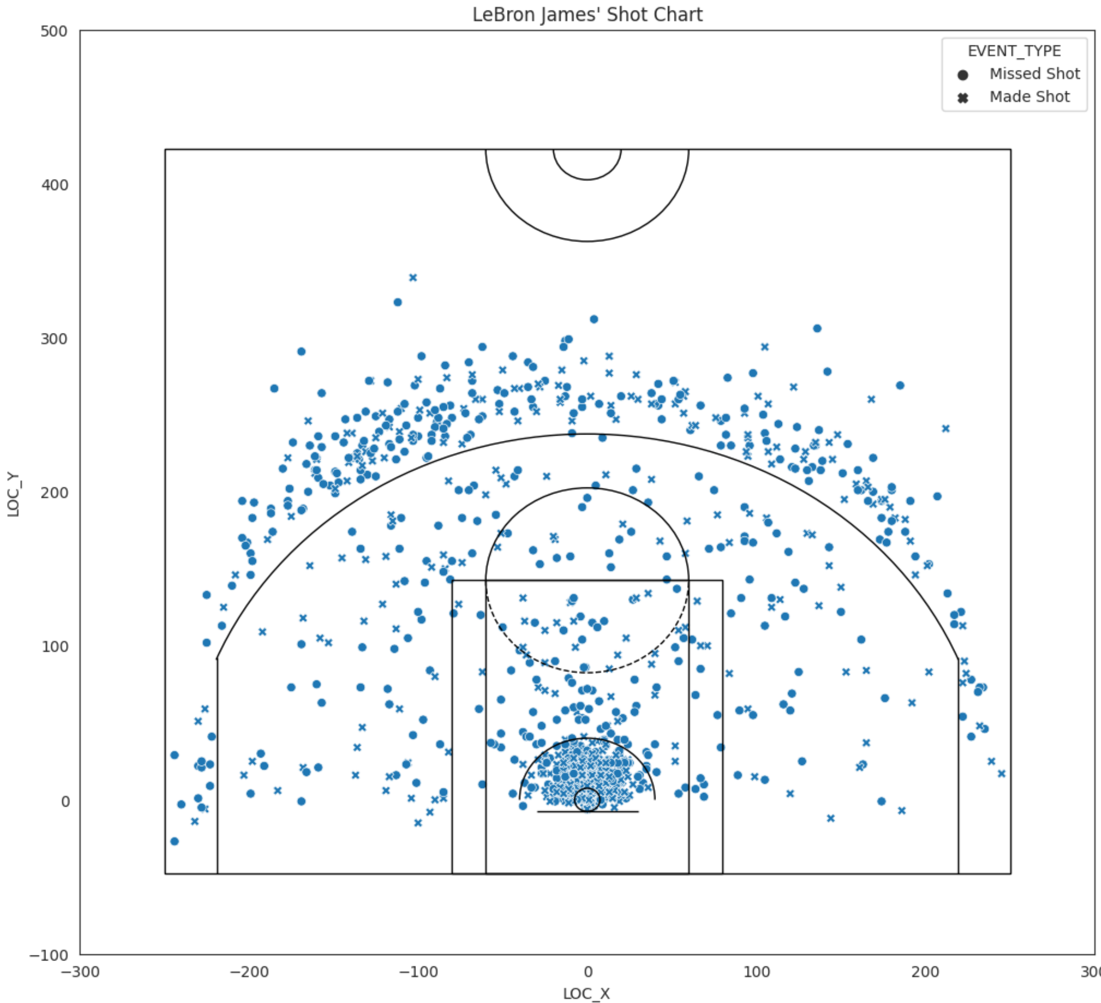
```
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```




```
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

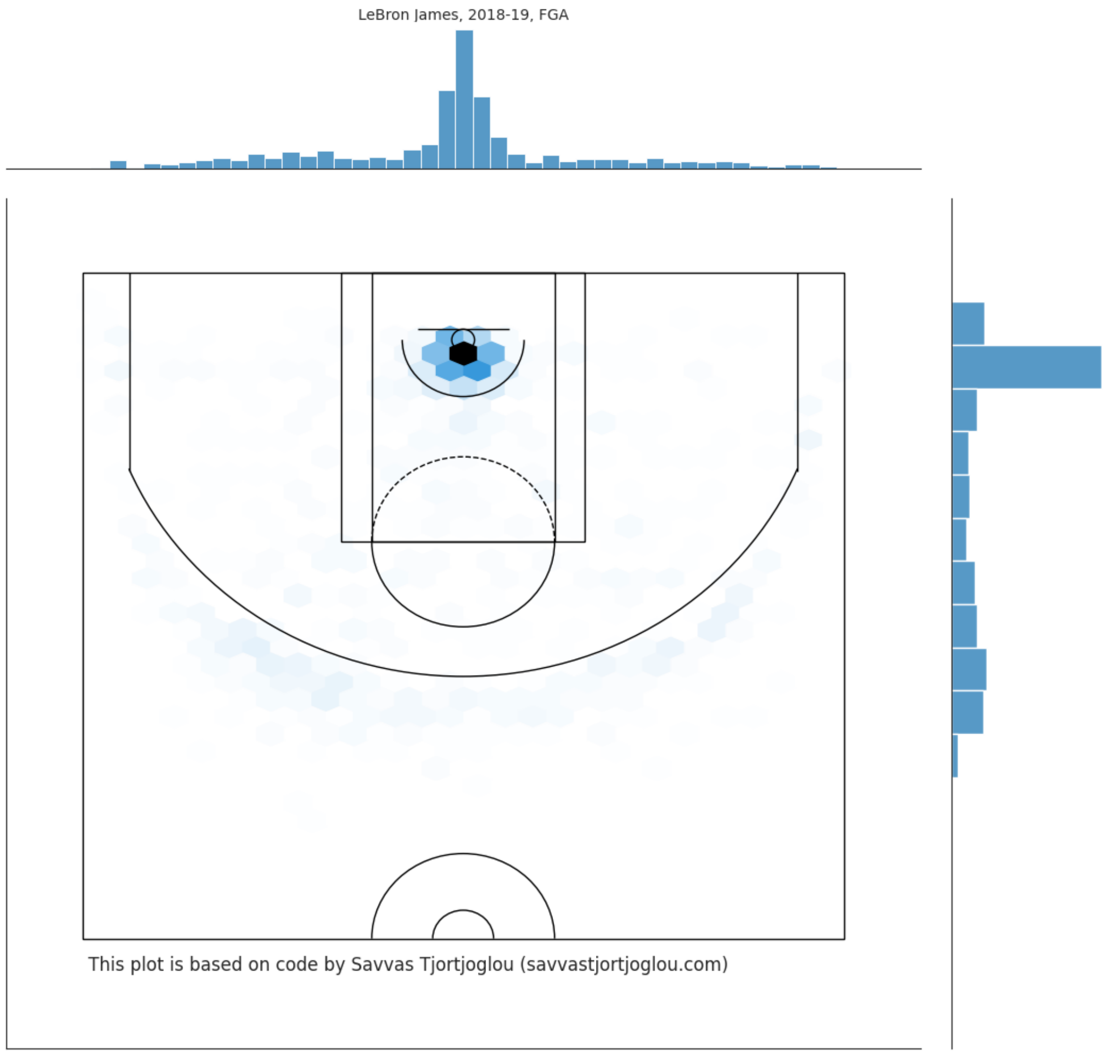
```
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```


Kyrie Irving, 2018-19, FGA
This plot is based on code by Savvas Tjortjoglou (savvastjortjoglou.com)




```
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1498: FutureWarning:
is_categorical_dtype is deprecated and will be removed in a future version. Use
isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Based off of these visualizations, we can see that both Kyrie Irving and LeBron James like to take shots from right around the net as would be normal in a lot of situations. But in addition to that Kyrie also takes a fairly balanced approach and shoots from all around the court especially 3 pointers. However, LeBron ventures out a lot less - seems that he's more aggressive and likes to sit right up against the net. Additionally, Kyrie favors his right side and takes opportunities to shoot from the space in between the arc and the lane lines.

Cell Intentionally Blank

1.5 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

Download the zip file and submit to Gradescope.

[39]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export(run_tests=True)
```

```
Running your submission against local test cases…



Your submission received the following results when run against available test
cases:



<IPython.core.display.HTML object>
```