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

Table of Contents

Assignment 3: Exploratory Data Analysis in Professional Basketball

In this assignment we'll conduct an exploratory data analysis of professional basketball data. 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.

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, 3). While these approaches will work when python is installed and running on your computer, NBA seems to block (pun intended) connections from Google Cloud where our course JupyterHub is running.

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 exclamation point in Jupyter notebook cell indicates that `bash` shell interpreter will execute your command.

```
wget -nc
https://ucsb.box.com/shared/static/z6y3etgikbzbnf0ld4brvc95xtgjcrie.zi
 -O nba-data.zip
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 directory named `data` (specified by `-d data`) and will overwrite any same filenames when extracting (specified by `-o`).

Following screencast videos show the terminal vs. Jupyter notebook's `!` exclamation way of running command line commands.

In [2]:
```
# Run your commands in this cell
```

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&ContextMeasure=FGA&DateFrom=&DateTo=&GameID=&GameSegment=&LastNG`

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`.

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

Question 1a: Team Data

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

In [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/
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
```

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

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.

In [6]:
```
allteams = pd.DataFrame(step_3, columns=step_4)
allteams = allteams[allteams['ABBREVIATION'].notna()]
```

In [7]:
```
grader.check("q1a")
```

Out[7]:

q1a passed! 🎉

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=0`. Assemble pandas DataFrame with name `allplayers`. Set row index to be `PERSON_ID` and `sort_index`.

In [8]:
```
dirname  = 'data' # directory_name
filename = 'commonallplayers?LeagueID=00&Season=2018-19&IsOnlyCurrentSeason=
step_1 = (Path(dirname) / filename).read_text()
step_2 = json.loads(step_1)                      # dict
step_3 = step_2['resultSets'][0]['rowSet']      # list
step_4 = step_2['resultSets'][0]['headers']     # list


allplayers = pd.DataFrame(step_3, columns=step_4)
allplayers.set_index('PERSON_ID', inplace=True)
allplayers.sort_index(inplace=True)

print(allplayers.head())
```

```
          DISPLAY_LAST_COMMA_FIRST DISPLAY_FIRST_LAST  ROSTERSTATUS FROM_YE
AR  \
PERSON_ID
2                     Scott, Byron        Byron Scott             0      19
83
3                      Long, Grant         Grant Long             0      19
88
7                     Schayes, Dan        Dan Schayes             0      19
81
9                  Threatt, Sedale     Sedale Threatt             0      19
83
12                     King, Chris         Chris King             0      19
93

          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

          TEAM_ABBREVIATION TEAM_CODE GAMES_PLAYED_FLAG  \
PERSON_ID
2                                                     Y
3                                                     Y
7                                                     Y
9                                                     Y
12                                                    Y

          OTHERLEAGUE_EXPERIENCE_CH
PERSON_ID
2                                00
3                                00
7                                00
9                                00
12                               00
```

In [9]:

```
grader.check("q1b")
```

Out[9]:

q1b passed! 🌟

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:

In [10]:
```
two_files = Path('data').glob('common*') # generator: https://www.educative.
list(two_files)                           # list
```

Out[10]:
```
[PosixPath('data/commonTeamYears?LeagueID=00&Season=2018-19'),
 PosixPath('data/commonallplayers?LeagueID=00&Season=2018-19&IsOnlyCurrentS
eason=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`.

In [11]:
```
allshots_files = list(Path('data').glob('shotchartdetail*'))
allshots_files.sort()

allshots_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']
    df = pd.DataFrame(step_3, columns=step_4)
    allshots_list.append(df)

allshots = pd.concat(allshots_list)
allshots.set_index('PLAYER_ID', inplace=True)
allshots.sort_index(inplace=True)

print(allshots.head())
```

```
                 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

              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

          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)

           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

          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

[5 rows x 23 columns]
```

In [12]:

```
grader.check("q1c")
```

Out[12]:

q1c passed! 🎉

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(????)
```

In [13]:
```
# 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_F
```

In [14]:
```
grader.check("q1d")
```

Out[14]:

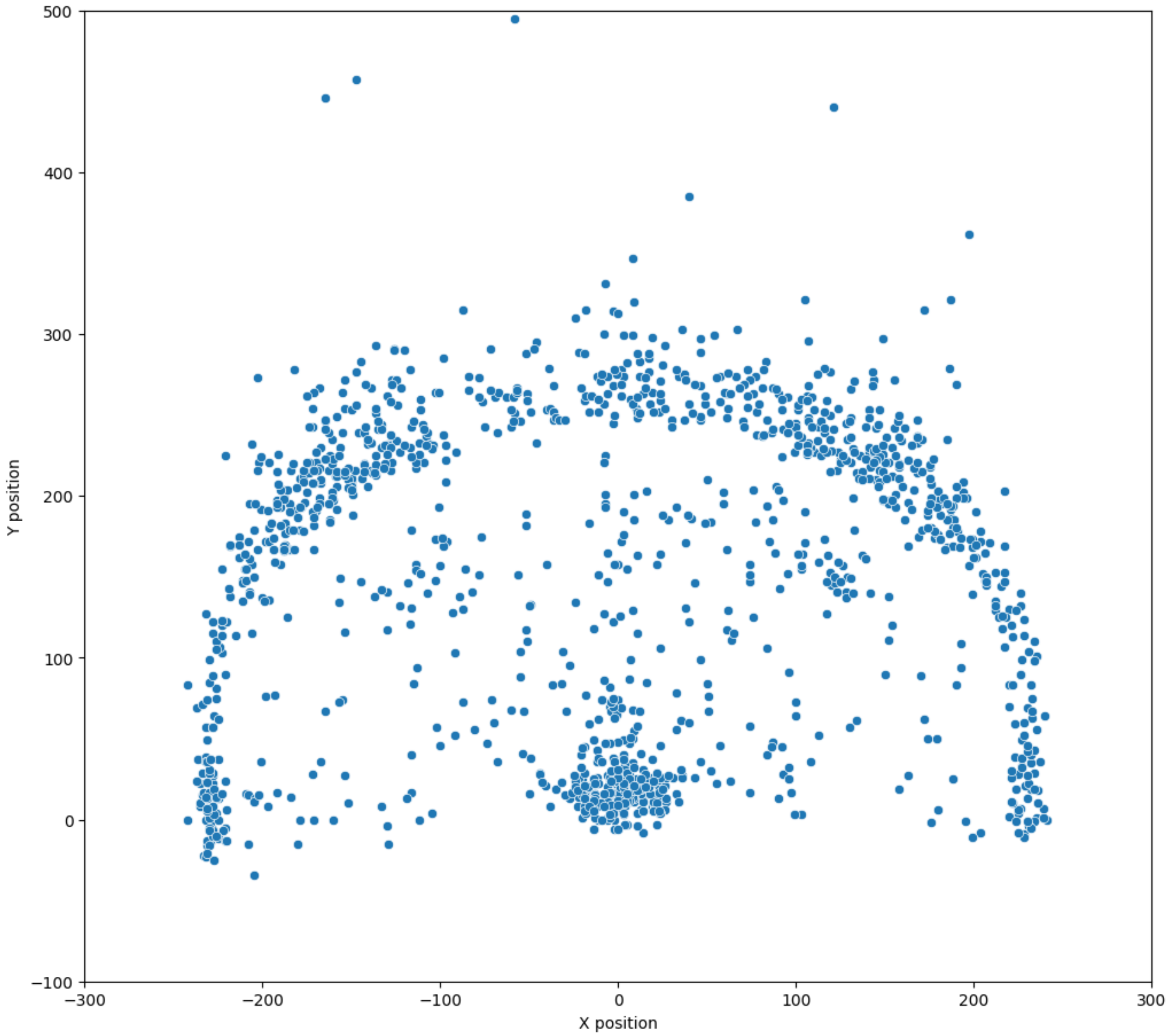
q1d passed! ✨

Question 2: Visualization

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 `ax2a`. We can set properties of the figure by calling methods on `ax2a`. Use this approach to set the x-axis limits to span (-300, 300), the y-axis limits to span (-100, 500).

In [15]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=[12, 11])
ax2a = sns.scatterplot(data=curry_data, x='LOC_X', y='LOC_Y')

# Set x/y limits and labels
ax2a.set_xlim(-300, 300)
ax2a.set_ylim(-100, 500)
ax2a.set_xlabel('X position')
ax2a.set_ylabel('Y position')

plt.show()
```




```
In [16]: grader.check("q2a")
```

Out[16]:

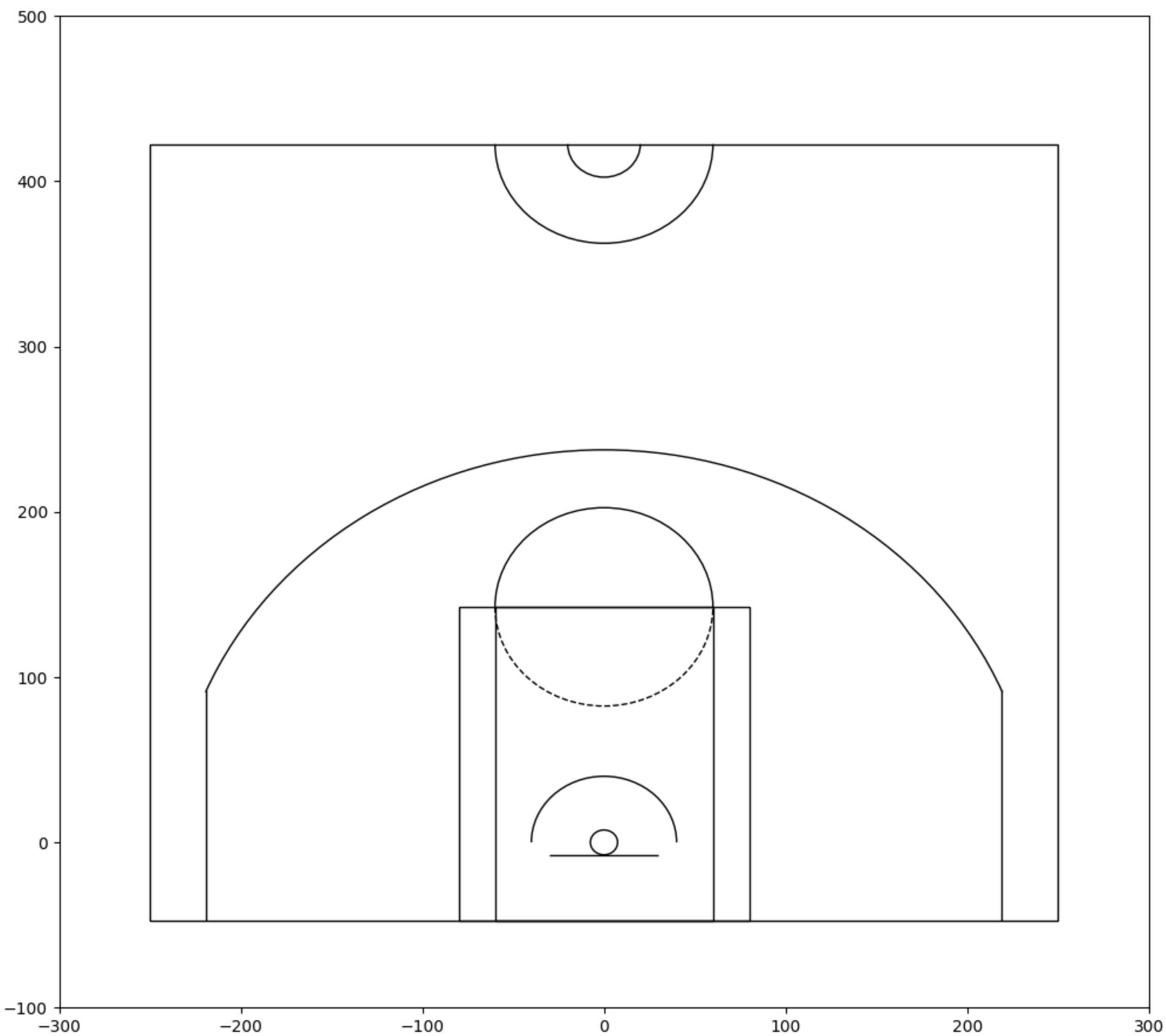
q2a passed! 🎉

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.

```
In [17]: ## 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
    from matplotlib.pyplot import gca

    # If an axes object isn't provided to plot onto, just get current one
```

```python
    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 0f 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=col
    # 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 th
    # threes
    three_arc = Arc((0, 0), 475, 475, theta1=22.5, theta2=157.5, linewidth=l
                    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]
```

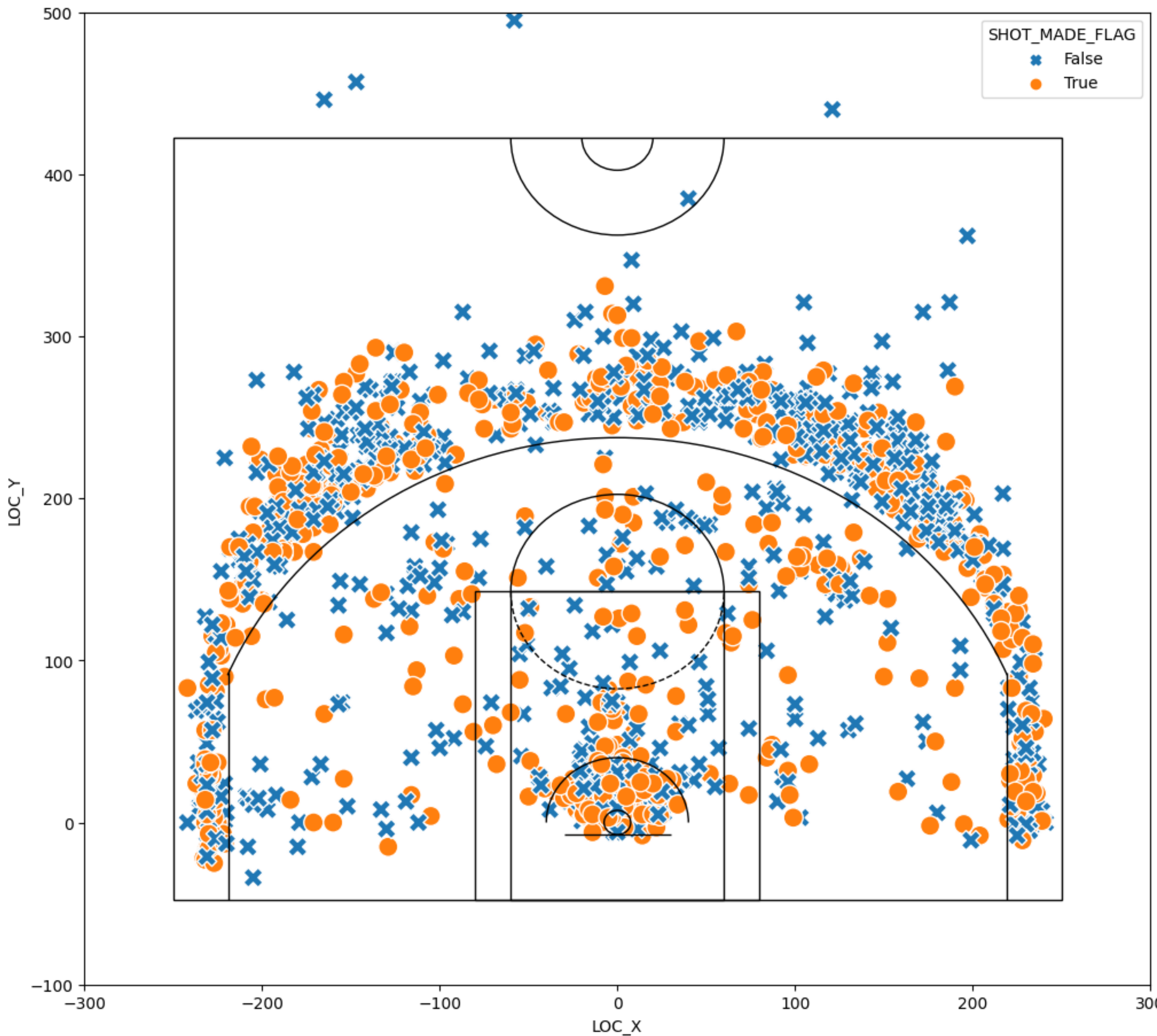
```
    if outer_lines:
        # Draw the half court line, baseline and side out bound lines
        outer_lines = Rectangle((-250, -47.5), 500, 470, linewidth=lw,
                                color=color, fill=False)
        court_elements.append(outer_lines)

    # 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()
```



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`.

In [18]:
```
plt.figure(figsize=(12, 11))
markers = {0 : "X", 1 : "o"}
ax = sns.scatterplot(x="LOC_X", y="LOC_Y", data=curry_data, hue="SHOT_MADE_F
draw_court(ax, outer_lines=True)
ax.set_xlim(-300, 300)
ax.set_ylim(-100, 500)
plt.show()
```



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?

An effective visualization is colorcoding where the shots are and using proper size. Relative to players in the league we can see Curry loves taking shots outside the 3 point area and we can color code which shots he made and which ones he didn't. We can improve this by comparing it to another player who is also a point guard.

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.

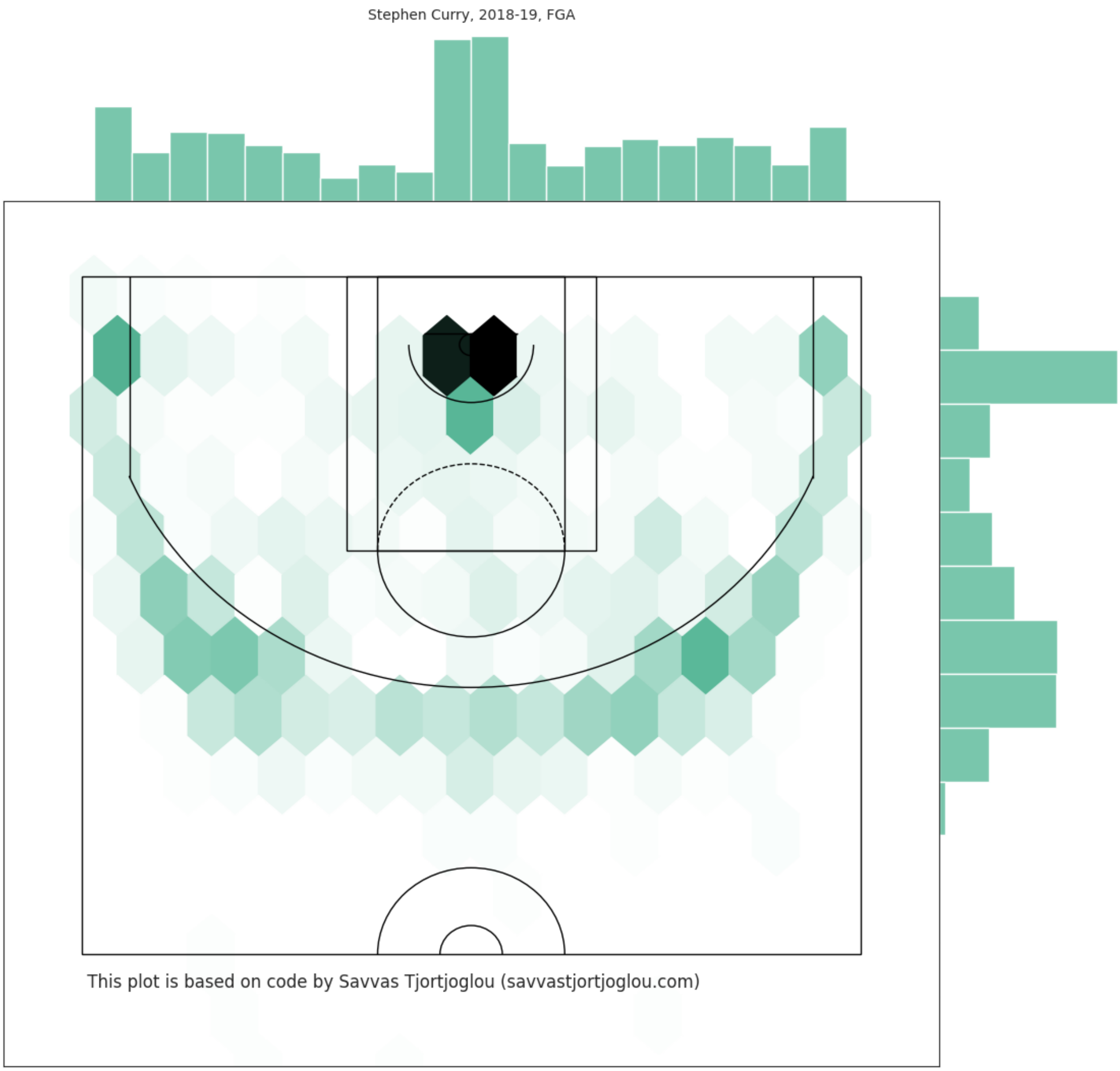
In [19]:
```
curry_data.reset_index()
```

Out[19]:

	PLAYER_ID	GRID_TYPE	GAME_ID	GAME_EVENT_ID	PLAYER_NAME	TEAM_ID	T
0	201939	Shot Chart Detail	0021800862	117	Stephen Curry	1610612744	
1	201939	Shot Chart Detail	0021800862	600	Stephen Curry	1610612744	
2	201939	Shot Chart Detail	0021800862	576	Stephen Curry	1610612744	
3	201939	Shot Chart Detail	0021800862	484	Stephen Curry	1610612744	
4	201939	Shot Chart Detail	0021800862	467	Stephen Curry	1610612744	
...	...	...	...	...	...	...	
1335	201939	Shot Chart Detail	0021800494	563	Stephen Curry	1610612744	
1336	201939	Shot Chart Detail	0021800494	510	Stephen Curry	1610612744	
1337	201939	Shot Chart Detail	0021800494	467	Stephen Curry	1610612744	
1338	201939	Shot Chart Detail	0021800494	447	Stephen Curry	1610612744	
1339	201939	Shot Chart Detail	0021800494	685	Stephen Curry	1610612744	

1340 rows × 24 columns

In [20]:
```
sns.set_style("white")
joint_shot_chart = sns.jointplot(data=curry_data, x='LOC_X', y='LOC_Y', kinc
                                 color='#4CB391', marginal_kws=dict(bins=20)
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 Scource and Author
ax.text(-250,445,'\n This plot is based on code by Savvas Tjortjoglou (savva
        fontsize=12);
```



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.

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.filters.gaussian_filter` on the result of `np.histogram2d` with smoothing parameter `sigma`. This will create a smoothed version of the raw data histograms.

In [21]:

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

    """Given data frame of shots, compute a 2d matrix of binned counts is co

    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

    xedges, yedges = bin_edges

    # Call np.histogram2d
    binned, _, _ = np.histogram2d(df['LOC_X'], df['LOC_Y'], bins=[xedges, ye

    if density:
        # Recompute 'binned' using "gaussian_filter"
        binned = ndimage.filters.gaussian_filter(binned, sigma=sigma)

        # Normalize the histogram to be a "density", e.g. mass across all bi
        binned /= np.sum(binned)

    return binned, xedges, yedges
```

In [22]:

```
grader.check("q3a")
```

Out[22]:

q3a passed! 🚀

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:

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

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

curry_binned_unsmoothed, xe, ye = bin_shots(curry_data, bin_edges, density=F
curry_binned_smoothed1,  xe, ye = bin_shots(curry_data, bin_edges, density=T
curry_binned_smoothed5,  xe, ye = bin_shots(curry_data, bin_edges, density=T
```

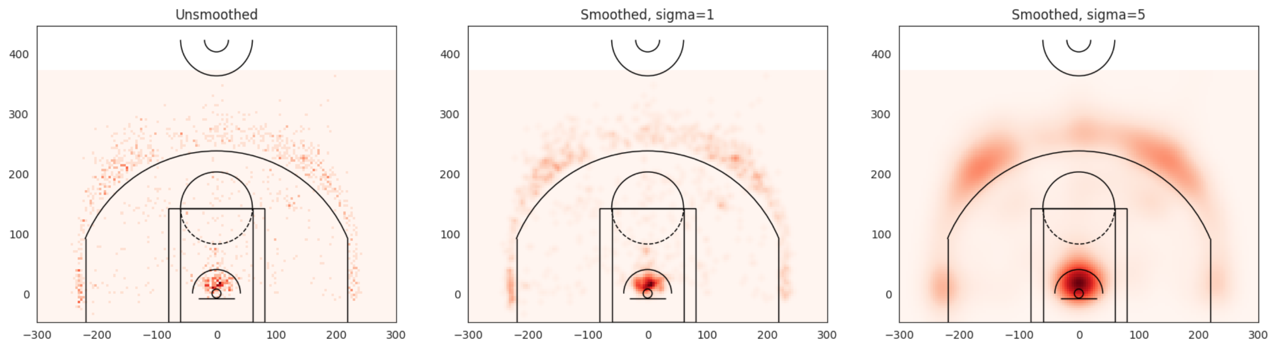
```
/tmp/ipykernel_104/455095783.py:25: DeprecationWarning: Please use `gaussia
n_filter` from the `scipy.ndimage` namespace, the `scipy.ndimage.filters` n
amespace is deprecated.
  binned = ndimage.filters.gaussian_filter(binned, sigma=sigma)
```

The function below can be used to visualize the shots as a heatmap:

In [25]:
```python
def plot_shotchart(binned_counts, xedges, yedges, ax=None, use_log=False, cm

    """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 [Fa
        cmap: Set the color map https://matplotlib.org/examples/color/colorm
    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)
```

Create 3 side by side plots of `curry_binned_unsmoothed`, `curry_binned_smoothed1` and `curry_binned_smoothed5`

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

plot_shotchart(curry_binned_unsmoothed, xedges, yedges, ax=ax[0], use_log=Fa
ax[0].set_title("Unsmoothed")

plot_shotchart(curry_binned_smoothed1, xedges, yedges, ax=ax[1], use_log=Fal
ax[1].set_title("Smoothed, sigma=1")

plot_shotchart(curry_binned_smoothed5, xedges, yedges, ax=ax[2], use_log=Fal
ax[2].set_title("Smoothed, sigma=5")

fig.show()
```



Vectorize Shot Images

- Here we proceed create a dictionary of smoothed patterns, each vectorized into a 1-d array.
- 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.

In [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), de
        tmp2, xedges, yedges = bin_shots(pdf, bin_edges=(xedges, yedges), de

        ## 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)
```

```
/tmp/ipykernel_104/455095783.py:25: DeprecationWarning: Please use `gaussia
n_filter` from the `scipy.ndimage` namespace, the `scipy.ndimage.filters` n
amespace is deprecated.
  binned = ndimage.filters.gaussian_filter(binned, sigma=sigma)
Number of shot regions (p): 15750
Number of players (n): 388
```

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 3b.

Bases matrix: W

Columns W_i contain the shot "bases". First, we will try it with $r = 3$ bins in 4a, and then with $r = 10$ bins in 4d.

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.

In [28]:
```python
## Non-negative Matrix Factorization
def non_negative_marix_decomp(n_components, array_data):
    import sklearn.decomposition as skld
    model = skld.NMF(n_components=n_components, init='nndsvda', max_iter=500
    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)
```

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).

In [29]:
```python
W3, H3 = non_negative_marix_decomp(n_components=3, array_data=X)
```

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

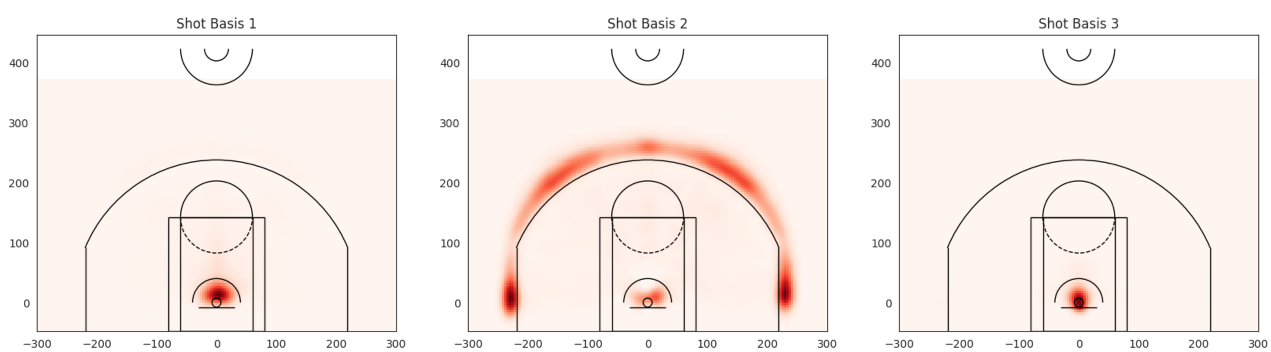
Out[30]:

q4a passed! 🌈

Question 4b: Visualizing Shot Types

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

In [31]:
```
def plot_vectorized_shotchart(vec_counts, xedges, yedges, ax=None, use_log=F
    """
    Plots 2d heatmap from vectorized heatmap counts

    Args:
        vec_counts (array-like): vectorized output of numpy.histogram2d
        xedges (array-like): bin edges in arrays
        yedges (array-like): bin edges in arrays
        ax (AxesSubplot, optional): figure axes. Defaults to None.
        use_log (bool, optional): whether to use logarithmic scaling. Defaul
        cmap (str, optional): name of the colormap. Defaults to 'Reds'.

    Returns:
        AxesSubplot: axes with plot
    """
    # xedges and yedges lengths
    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_l


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

## Write a for loop
for i in range(3):
    plot_vectorized_shotchart(W3[:,i], xedges, yedges, ax=ax[i], use_log= Tr
    ax[i].set_title('Shot Basis %i' % (i+1))

plt.show()
```



Question 4c: Reconstruction Error

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

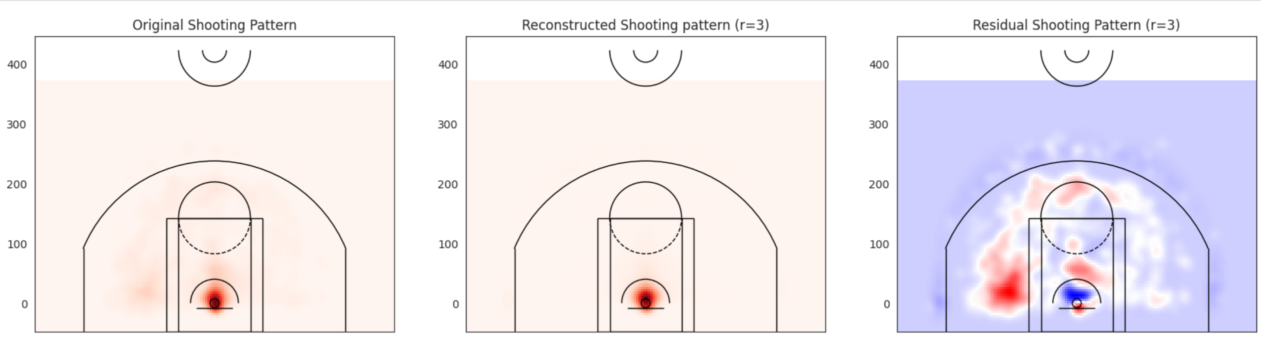
$$\hat{X} = WH$$

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

In [32]:
```
X3_hat = W3 @ H3
```

Plot X, $\hat{X}$ and the residual ($X - \hat{X}$) for the player named LaMarcus Aldridge. Remember, each column of X is a vectorized matrix corresponding to the binned (or smoothed binned) shot information.

In [33]:
```
# Find the player_id of LaMarcus Aldridge
query_str = 'DISPLAY_FIRST_LAST == "LaMarcus Aldridge"'
player_id = allplayers.query(query_str).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, ye
original_shotchart.set_title('Original Shooting Pattern')

reconstructed_shotchart = plot_vectorized_shotchart(X3_hat[:, to_plot_idx],
reconstructed_shotchart.set_title('Reconstructed Shooting pattern (r=3)')

residual_chart = plot_vectorized_shotchart(X[:, to_plot_idx] - X3_hat[:, to_
residual_chart.set_title('Residual Shooting Pattern (r=3)')

fig.show()
```



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.

The sequential pallete allows us to see where the highest concentration of shots are and the diverging pallet allows us to see a difference in where shots are made in both players.

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.

In the reconstructed are the player likes to shoot more in the paint closer to the net, while in the relative area it is more outside the paint but inside the three point area.

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`.

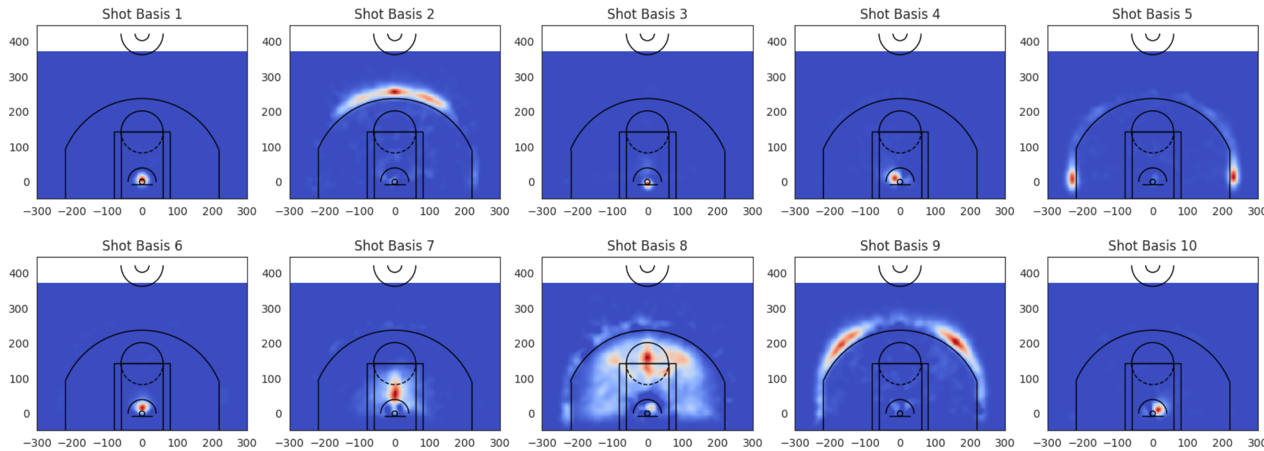
Hint: Study the following code

```
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()
```

In [39]:

```
import sklearn.decomposition as skld
model10 = skld.NMF(n_components=10, init='random', random_state=0)
W10 = model10.fit_transform(X)
H10 = model10.components_

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

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

```
/opt/conda/lib/python3.10/site-packages/sklearn/decomposition/_nmf.py:1665:
ConvergenceWarning: Maximum number of iterations 200 reached. Increase it t
o improve convergence.
  warnings.warn(
```

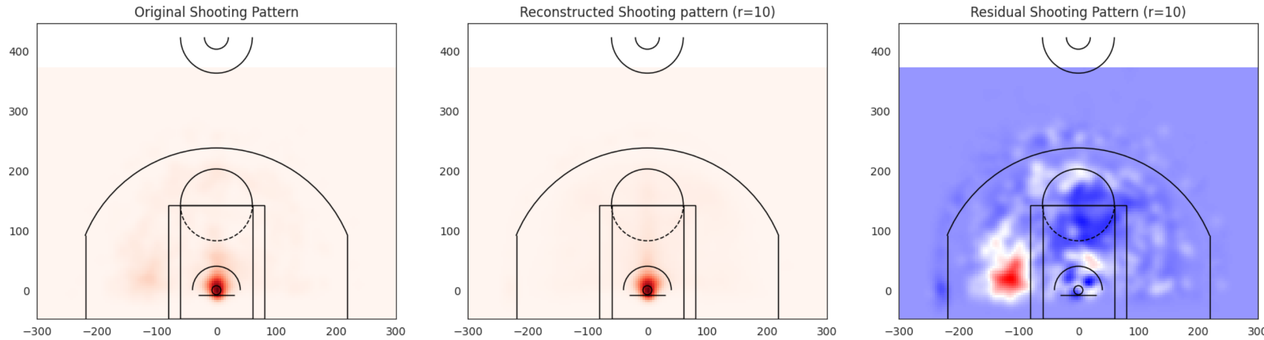


If you did things correctly, you should be really impressed! We've identified potentially interesting patterns of shooting styles without actually specifying anything about the way basketball is played or where the relevant lines are on the court. The resulting images are based only on the actual behavior 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.

Some players like shot 4 and shot 10 have more preferences of being right or left hand dominant. However, most players seem to be rather symmetric with their shooting

Repeat part 4b, and again plot original, reconstructed and residual shot chats for LaMarcus Aldridge.

In [40]:
```python
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)
original_shotchart = plot_vectorized_shotchart(X[:, to_plot_idx], xedges, ye
reconstructed_shotchart = plot_vectorized_shotchart(X10_hat[:, to_plot_idx],
residual_chart = plot_vectorized_shotchart(X[:, to_plot_idx] - X10_hat[:, to

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)');
plt.show()
```



Question 4f: Comparing Players

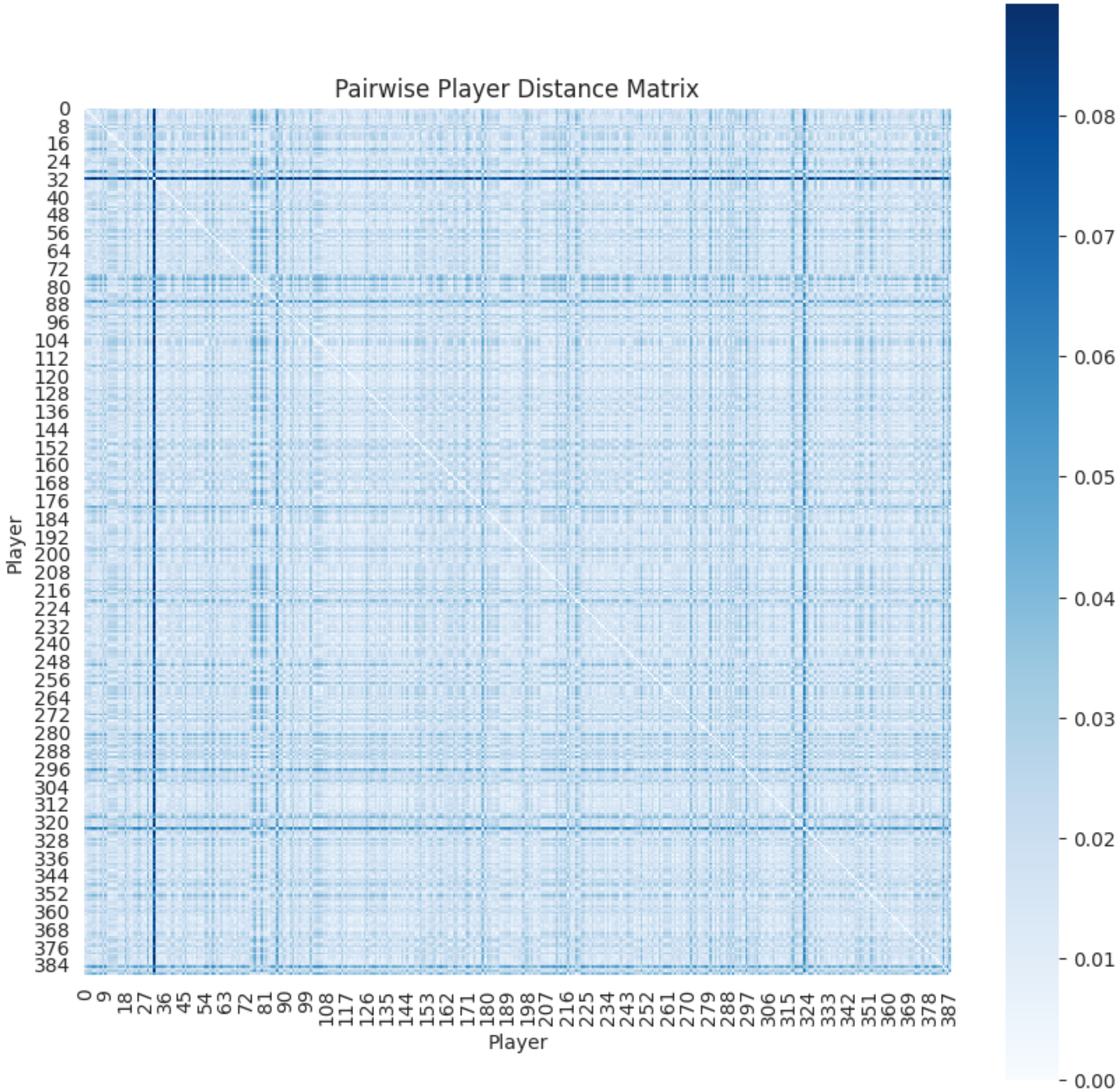
With `H10` matrix, it is possible to compare any pair of players. For all players pairwise, i and j, compare using euclidean distance between their coefficients:

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

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

In [41]:

```
from scipy.spatial.distance import pdist, squareform
import seaborn as sns

# Compute pairwise distances between players
distances = pdist(H10.T, metric='euclidean')

# Convert distances to a square matrix
dist_matrix = squareform(distances)

# Create heatmap of player distance matrix
fig, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(dist_matrix, cmap='Blues', square=True, ax=ax)
ax.set_xlabel('Player')
ax.set_ylabel('Player')
ax.set_title('Pairwise Player Distance Matrix')
plt.show()
```

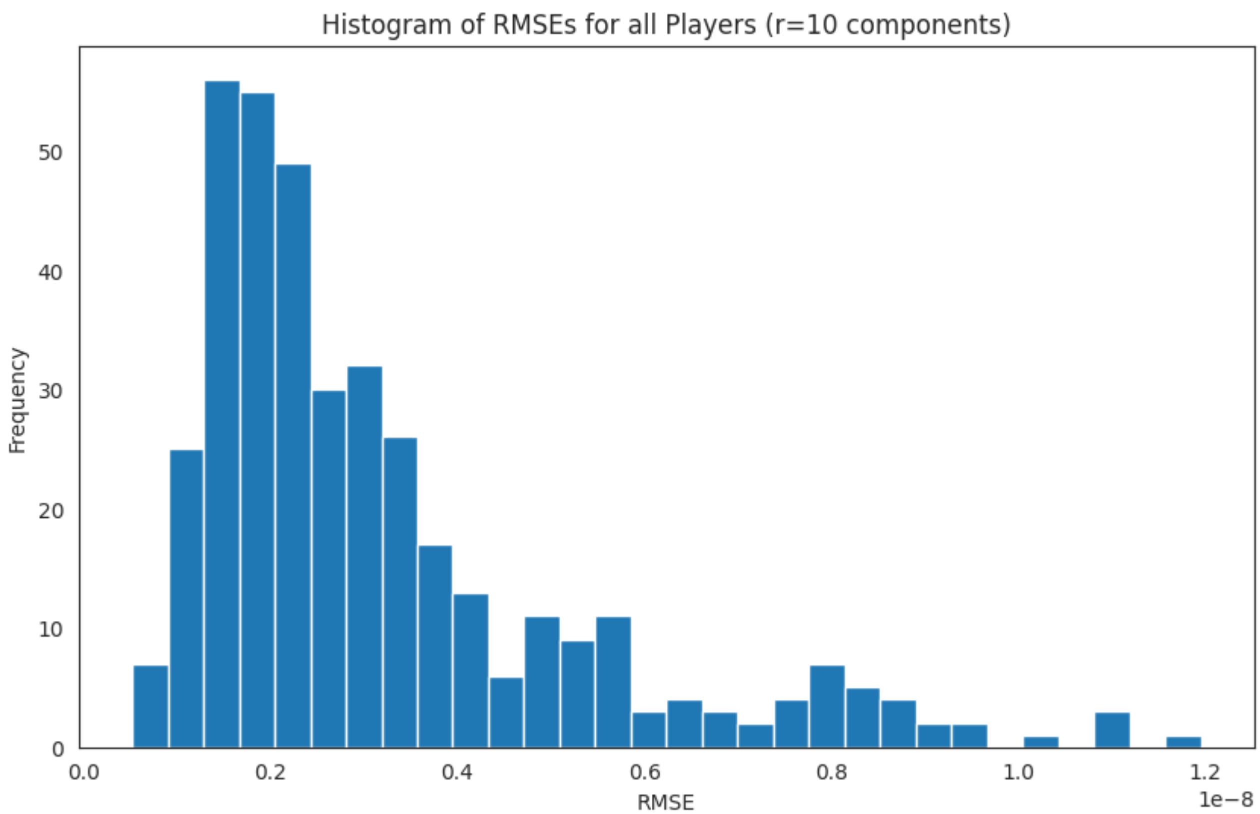


The pairwise distance is the smallest between The pairwise distance is the largest between player 28 and 32

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).

In [42]:
```
from sklearn.metrics import mean_squared_error

# Compute X10_hat for each player


# Compute residuals and RMSEs for each player
rmse = np.mean(np.square(X - X10_hat), axis = 0)

# Plot histogram of RMSEs
fig, ax = plt.subplots(figsize=(10,6))
ax.hist(rmse, bins=30)
ax.set_xlabel('RMSE')
ax.set_ylabel('Frequency')
ax.set_title('Histogram of RMSEs for all Players (r=10 components)')
plt.show()
```



The distribution of this graph is skewed to the right with most of the RMSE being at 0, however some rmse go closer to 1.2*10^-8. This means that there is not much error in the grap

Cell Intentionally Blank

To double-check your work, the cell below will rerun all of the autograder tests.

In [43]:
```
grader.check_all()
```

```
Out[43]: q1a results: All test cases passed!

q1b results: All test cases passed!

q1c results: All test cases passed!

q1d results: All test cases passed!

q2a results: All test cases passed!

q3a results: All test cases passed!

q4a results: All test cases passed!
```

Submission

1. Save file to confirm all changes are on disk
2. Run *Kernel > Restart & Run All* to execute all code from top to bottom
3. Save file again to write any new output to disk
4. Select *File > Save and export Notebook as > HTML*.
5. Open in Google Chrome and print to PDF.
6. Submit to Gradescope