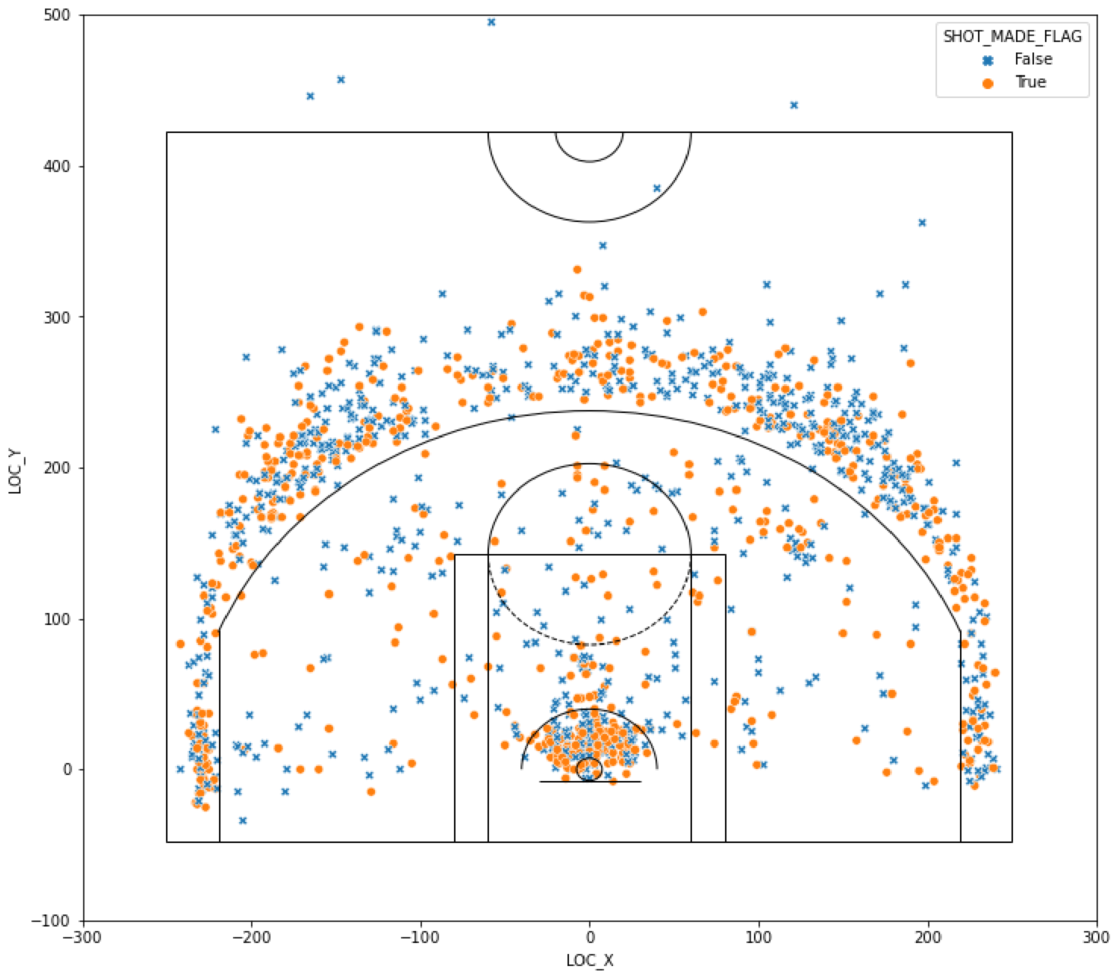
0.0.1 Question 2b: All Shots Scatter Plot + Court Outline

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(data = curry_data, x ='LOC_X', y = 'LOC_Y', markers = markers,
                              hue = 'SHOT_MADE_FLAG', style = 'SHOT_MADE_FLAG')

         draw_court(outer_lines=True)

         plt.xlim(-300,300)
         plt.ylim(-100,500)
         plt.show()
```



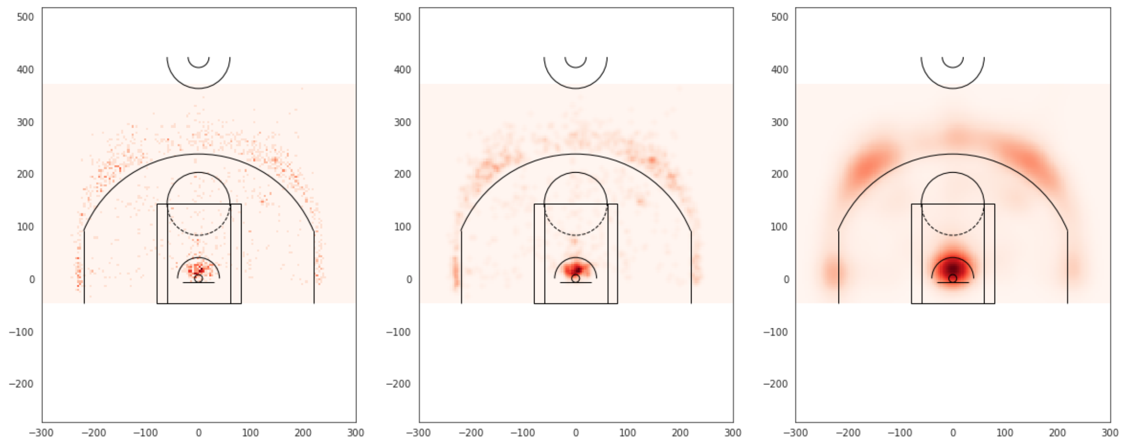
0.0.2 Question 2c: Analyzing the Visualization

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?

For the purpose of understanding the types of shots that Stephen Curry likes to take and is good at taking, relative to other players in the league, this is an ineffective visualization.

This visualization has no information on other players in the league so comparison is impossible. Further, overlapping dots make it quite difficult to discern the quantity of datapoints in a given location. Thus, it is difficult to infer where the most shots were taken from. It seems like Steph took more shots from 3 but the solid region beneath the basket makes it difficult to be sure.

```
In [27]: fig, ax = plt.subplots(1, 3, figsize=(20,60))

         plot_shotchart(curry_binned_unsmoothed, xe, ye, ax = ax[0])

         plot_shotchart(curry_binned_smoothed1, xe, ye, ax = ax[1])

         plot_shotchart(curry_binned_smoothed5, xe, ye, ax = ax[2])

         fig.show()
```



0.0.3 Question 4b: Visualizing Shot Types

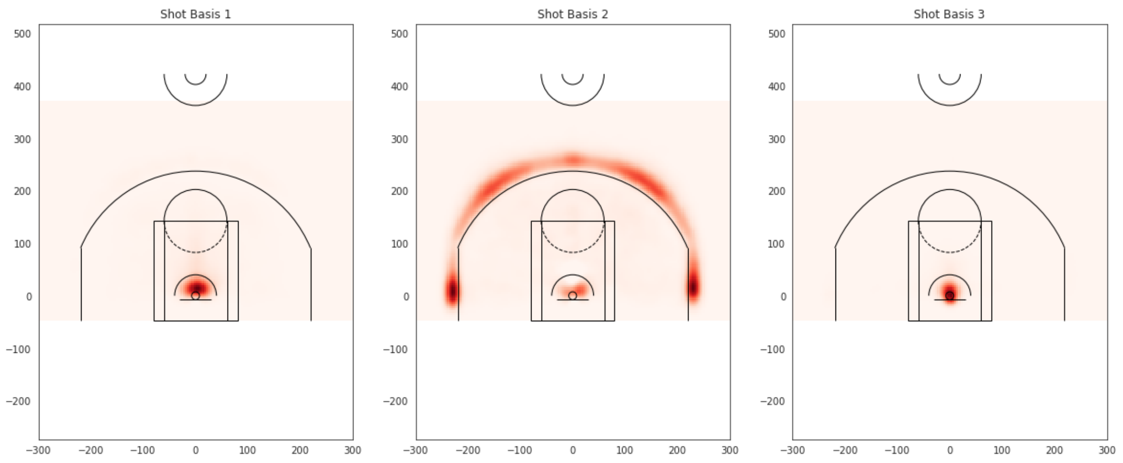
Plot the first three basis images by calling `plot_vectorized_shot_chart` below on the columns of `W3`.

```
In [36]: print(type(W3))
         np.vectorize(np.histogram2d(W3[:,1], W3[:,2]))
```

```
<class 'numpy.ndarray'>
```

```
Out[36]: <numpy.vectorize at 0x7fd3c495b430>
```

```
In [37]: 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 = np.reshape(vec_counts, (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):
             plot_vectorized_shotchart(W3[:,i], xe, ye, ax = ax[i])
             ax[i].set_title('Shot Basis %i' % (i+1))
```


Shot Basis 1
Shot Basis 2
Shot Basis 3
500
400
300
200
100
0
−100
−200
−300
−200
−100
0
100
200
300

0.0.4 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 [38]: X3_hat = W3 @ H3
         #print(allplayers)
         player_id = allplayers[allplayers['DISPLAY_FIRST_LAST'] == 'LaMarcus Aldridge']
         print(player_id.index[0])

         print(X.shape)
         print(X)
```

```
200746
(15750, 388)
[[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.]]
```

0.0.5 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.

It makes sense to use a sequential palette here because the shot location data is non-negative and ordered. Sequential palettes should be used for representing information that has ordering, so this makes sense.

It makes sense to use a diverging palette here because the shot location data is centered around zero. Diverging palettes should be used for representing information that has a critical middle value, so this makes sense.

0.0.6 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`.

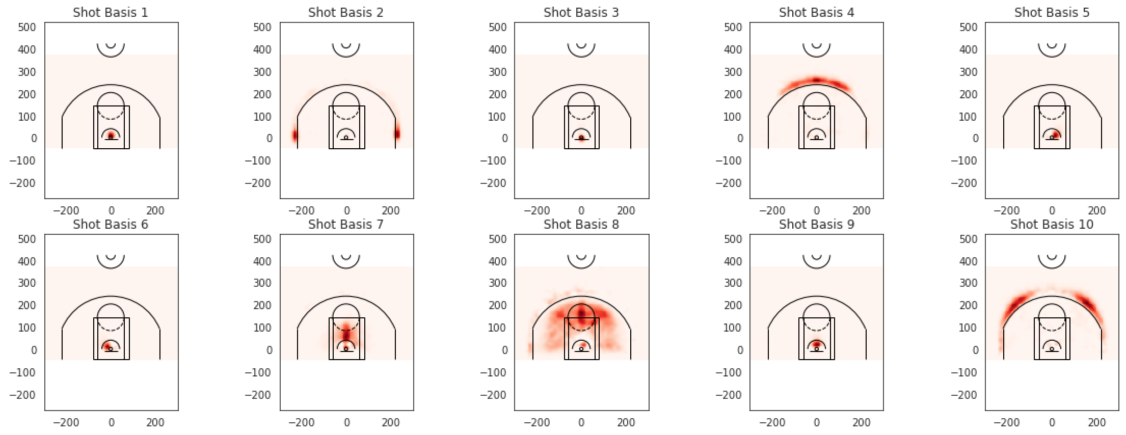
Hint: Study the following code

```
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 [40]: W10, H10 = non_negative_marix_decomp(10, X)
         print(W10)
```

```
[[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.]]
```

```
In [41]: 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], xe, ye, ax = ax[i])
             ax[i].set_title('Shot Basis %i' % (i+1))
             #ax[i//5, i % 5].set_title('Shot Basis %i' % (i+1))
```

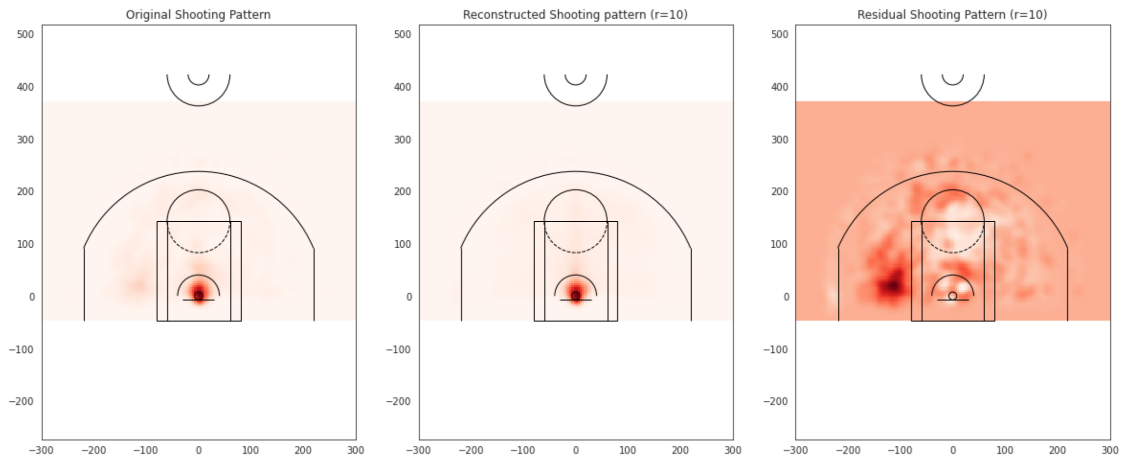

Shot Basis 1
Shot Basis 2
Shot Basis 3
Shot Basis 4
Shot Basis 5
Shot Basis 6
Shot Basis 7
Shot Basis 8
Shot Basis 9
Shot Basis 10
500
400
300
200
100
0
−100
−200
−200
0
200

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.

Yes, there is evidence of asymmetry in shooting behavior. Consider Shot Basis 5 and Shot Basis 6. These are very similar, Shot Basis 5 is right of the basket while Shot Basis 6 is left of the basket. The existence of asymmetric shot basis implies that shooting behavior is also asymmetric.

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

```
In [42]: 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, yedges, ax = ax[0])
         reconstructed_shotchart = plot_vectorized_shotchart(X10_hat[:, to_plot_idx], xedges, yedges, a
         residual_chart = plot_vectorized_shotchart((X[:, to_plot_idx] - X10_hat[:, to_plot_idx]), xedg

         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)');
```



0.0.7 Question 4f: Comparing Players

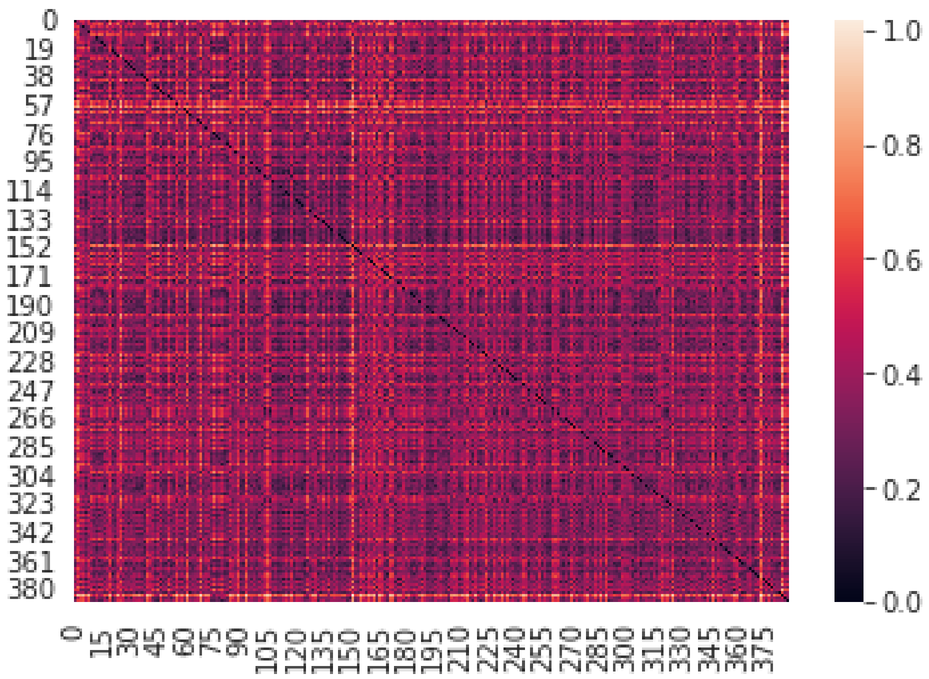
With `H10` matrix, it is possible to compare any pair of players. For all players pairwise, i and j, compare using euclidean distance between their coefficients:

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

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 [99]: #print(H10)
         #print(H10.shape)
         #print(H10[:,0])

         zeros = np.zeros((388,388))

         for i in range(388):
             for j in range(388):
                 zeros[i][j] = np.linalg.norm(H10[:,i] - H10[:,j])

         zeros
         sns.heatmap(zeros)
```

```
Out[99]: <AxesSubplot:>
```



```
In [100]: np.fill_diagonal(zeros, np.nan)

          print(np.where(zeros == np.nanmax(zeros)))
          #print(pids[31])
          #print(pids[150])
          #print(allplayers)
          print('Max Distance')
          print(allplayers[allplayers.index == 203124]['DISPLAY_FIRST_LAST'])
          print(allplayers[allplayers.index == 200782]['DISPLAY_FIRST_LAST'])


          print(np.where(zeros == np.nanmin(zeros)))
          #print(pids[158])
          #print(pids[272])
          print('Min Distance')
          print(allplayers[allplayers.index == 203468]['DISPLAY_FIRST_LAST'])
          print(allplayers[allplayers.index == 1627750]['DISPLAY_FIRST_LAST'])
```

```
(array([ 31, 150]), array([150,  31]))
Max Distance
PERSON_ID
203124    Kyle O'Quinn
Name: DISPLAY_FIRST_LAST, dtype: object
PERSON_ID
```

```
200782    PJ Tucker
Name: DISPLAY_FIRST_LAST, dtype: object
(array([158, 272]), array([272, 158]))
Min Distance
PERSON_ID
203468    CJ McCollum
Name: DISPLAY_FIRST_LAST, dtype: object
PERSON_ID
1627750    Jamal Murray
Name: DISPLAY_FIRST_LAST, dtype: object
```

The players with the largest euclidean distance are Kyle O'Quinn and PJ Tucker. This implies these players are very different

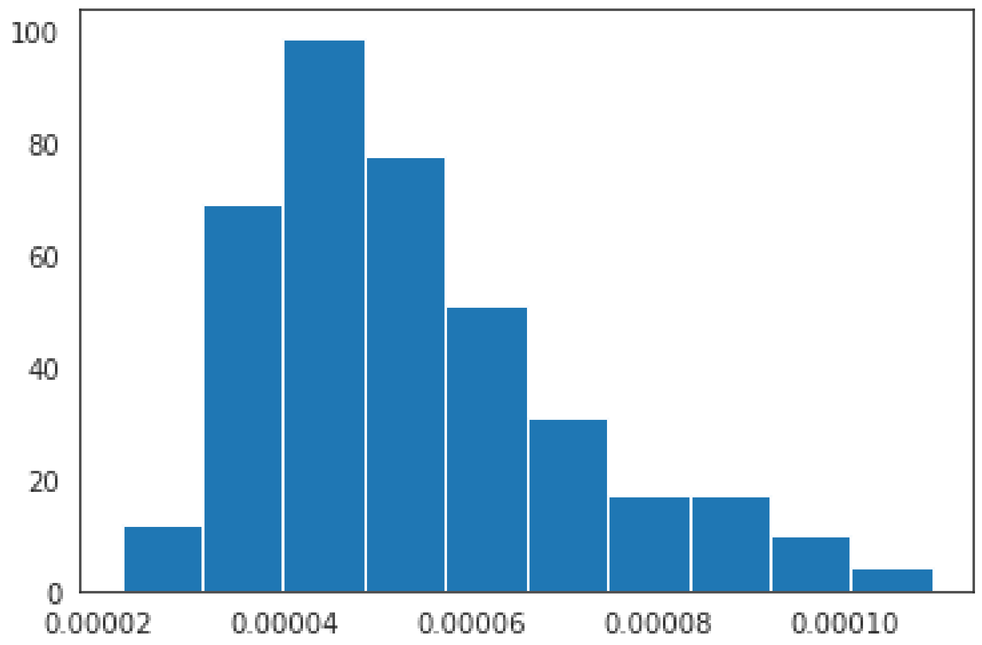
The players with the smallest euclidean distance are CJ McCollum and Jamal Murray. This implies these players are very similar.

0.0.8 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 [120]: zeros2 = np.zeros((388))

          #print(X10_hat.shape)
          #print(X.shape)

          for i in range(388):
              zeros2[i] = np.sqrt(np.mean((X10_hat[:,i]-X[:,i])**2))

          plt.hist(zeros2)

          print(max(zeros2))
          print(min(zeros2))
          print(np.where(zeros2 == np.nanmin(zeros2)))
          print(np.where(zeros2 == np.nanmax(zeros2)))
          print(pids[227])
          print(pids[128])
          print(allplayers[allplayers.index == 1626149]['DISPLAY_FIRST_LAST'])
          print(allplayers[allplayers.index == 202954]['DISPLAY_FIRST_LAST'])
          print(np.mean(zeros2))
```

```
0.00010946061707727095
2.244235831256383e-05
(array([227]),)
(array([128]),)
1626149
202954
PERSON_ID
1626149    Montrezl Harrell
Name: DISPLAY_FIRST_LAST, dtype: object
PERSON_ID
202954    Brad Wanamaker
Name: DISPLAY_FIRST_LAST, dtype: object
5.3757857618528786e-05
```



The player with the lowest RMSE is Montrezl Harrell. RMSE ≈ 2.244e-05

The player with the highest RMSE is Brad Wanamaker. RMSE = 0.00010946061707727095

The above distribution of RMSE is right skewed. The mean RMSE ≈ 5.376e-05, implying that our NMF is quite accurate and differed minutely from the true matrix X.

0.0.9 Question 4h: Proposing improvements

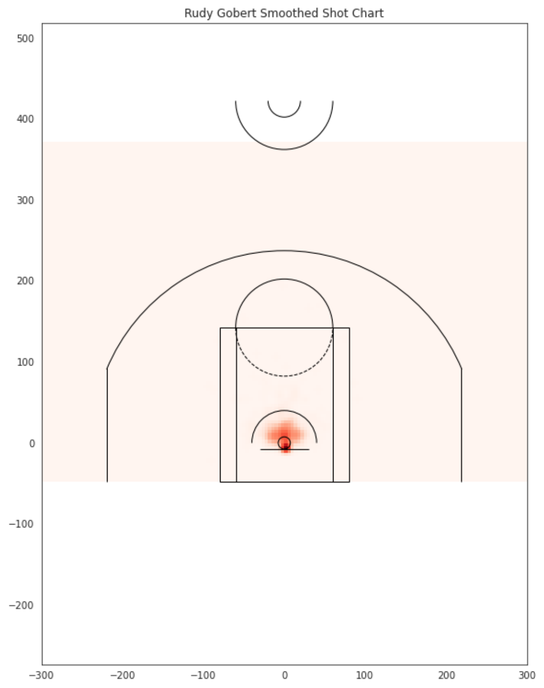
One of the main purposes of exploratory data analysis is to generate new ideas, directions, and hypothesis for future analyses and experiments. Take two players of your choice and compare their shooting patterns with various visualizations.

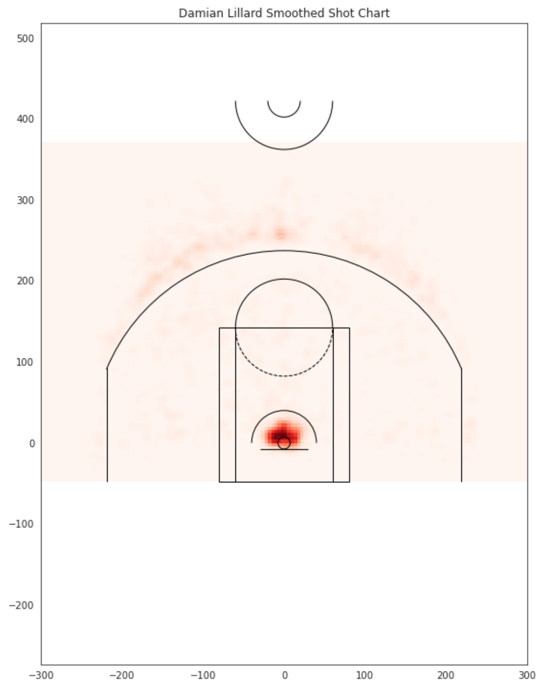
State any insights and defend your conclusions with visual and/or numerical comparisons.

```
In [183]: #print(allplayers.columns)
          #print(allplayers[allplayers['DISPLAY_FIRST_LAST']=='Rudy Gobert']['DISPLAY_FIRST_LAST'])
          # Above line finds Ruby's PID - 203497
          #print(allplayers[allplayers['DISPLAY_FIRST_LAST']=='Damian Lillard']['DISPLAY_FIRST_LAST'])
          # Above line finds Damian's PID - 203081

          rudy_data = allshots.query('PLAYER_ID == ' + '203497').astype({'SHOT_MADE_FLAG' : 'bool'})
          damian_data = allshots.query('PLAYER_ID == ' + '203081').astype({'SHOT_MADE_FLAG' : 'bool'})

          rudy_binned_smoothed,  xe, ye = bin_shots(rudy_data, bin_edges, sigma = 1, density = True)
          damian_binned_smoothed,  xe, ye = bin_shots(damian_data, bin_edges, sigma = 1, density = True

          fig, ax = plt.subplots(1, 2, figsize=(20,60))

          plot_shotchart(rudy_binned_smoothed, xe, ye, ax = ax[0])
          ax[0].set_title('Rudy Gobert Smoothed Shot Chart')

          plot_shotchart(damian_binned_smoothed, xe, ye, ax = ax[1])
          ax[1].set_title('Damian Lillard Smoothed Shot Chart')

          fig.show()

          print("Rudy's Shot Percentage: ",np.mean(rudy_data['SHOT_MADE_FLAG']))
          print("Damian's Shot Percentage: ",np.mean(damian_data['SHOT_MADE_FLAG']))

          print("Rudy's Avg. Shot Distance: ",np.mean(rudy_data['SHOT_DISTANCE']))
          print("Damian's Avg. Shot Distance: ",np.mean(damian_data['SHOT_DISTANCE']))

          print('Rudy has attempted', rudy_data.shape[0], 'shots.')
          print('Damian has attempted', damian_data.shape[0], 'shots.')

          #print(rudy_data.columns)
          damian2 = damian_data[damian_data['SHOT_DISTANCE'] < 1.74]
          print("Damian's Shot Percentage from Close: ",np.mean(damian2['SHOT_MADE_FLAG']))

          rudy_loc = np.where(pd.DataFrame(pids) == 203497)
          damian_loc = np.where(pd.DataFrame(pids) == 203081)
          euc_rudy_damian = zeros[rudy_loc, damian_loc]
          mean_euc = np.nanmean(zeros)
          print("Rudy and Damian's Euclidean Distance: ", euc_rudy_damian[0][0])
          #plt.hist(zeros.reshape(388 * 2));
          print('Average Euclidean Distance: ',np.nanmean(zeros))
```

```
#joint_shot_chart = sns.jointplot(data=rudy_data.reset_index(), x = 'LOC_X', y='LOC_Y', kind=
#                                  hue_order = 'SHOT_MADE_FLAG', ax = ax[2])
```

```
Rudy's Shot Percentage:  0.6685393258426966
Damian's Shot Percentage:  0.44422700587084146
Rudy's Avg. Shot Distance:  1.7429775280898876
Damian's Avg. Shot Distance:  15.158512720156557
Rudy has attempted 712 shots.
Damian has attempted 1533 shots.
Damian's Shot Percentage from Close:  0.6309523809523809
Rudy and Damian's Euclidean Distance:  0.47399500701472796
Average Euclidean Distance:  0.37417377292479875
```





Consider the above Shot Charts comparing Rudy Gobert and Damian Lilliard.

The charts imply these are very different players, primarily in that Damian Lilliard shoots more 3's than Rudy Gobert. This is confirmed by the allshots data, with Damian Lillard's average shot being taken from about 15 times further than Rudy Gobert's average shot.

The dark region under the basket seems to imply that Damian Lilliard is also more accurate from close, but this is not the case. Even when only considering Damian Lilliard's shots that are closer than Rudy Gobert's average shot, Damian Lilliard makes about 63% while Ruby Gobert's overall average is about 67%.

The aforementioned differences in shot distance and percentage imply that Rudy Gobert and Damian Lilliard are more dissimilar than average. This is confirmed by the Euclidean distance of their coefficients in the H10 matrix. Their Euclidean distance is about .473 whereas the average is about .374.

Overall, Damian Lillard and Rudy Gobert are quite dissimilar. Damian Lillard finds success from 3 while Rudy Gobert lives under the basket.