

Data Visualization

Density estimation

June 13th, 2022

New dataset - Stephen Curry's shots

Created dataset of shot attempts by the Stephen Curry in 2021-2022 season using `nbastatR`

```
library(tidyverse)
curry_shots <-
  read_csv("http://www.stat.cmu.edu/cmsac/sure/2022/materials/data/sports/xy_examples/curry_2022_")
head(curry_shots)
```

```
## # A tibble: 6 × 8
##   shot_x shot_y shot_distance is_shot_made period fg_type      shot_...1 shot_...2
##   <dbl> <dbl>         <dbl> <lgl>         <dbl> <chr>      <chr>      <chr>
## 1   -109    260           28 FALSE         1 3PT Field Goal Above ... Pullup...
## 2     48    257           26 FALSE         1 3PT Field Goal Above ... Runnin...
## 3   -165    189           25 TRUE          1 3PT Field Goal Above ... Jump S...
## 4    -13     12            1 FALSE         1 2PT Field Goal Restri... Drivin...
## 5    -15     22            2 FALSE         1 2PT Field Goal Restri... Layup ...
## 6     18     16            2 FALSE         1 2PT Field Goal Restri... Drivin...
## # ... with abbreviated variable names 1shot_zone, 2shot_type
```

- each row / observation is a shot attempt by Curry in the 2021 season
- **Categorical** / qualitative variables: `is_shot_made`, `fg_type`, `shot_zone`, `shot_type`
- **Continuous** / quantitative variables: `shot_x`, `shot_y`, `shot_distance`

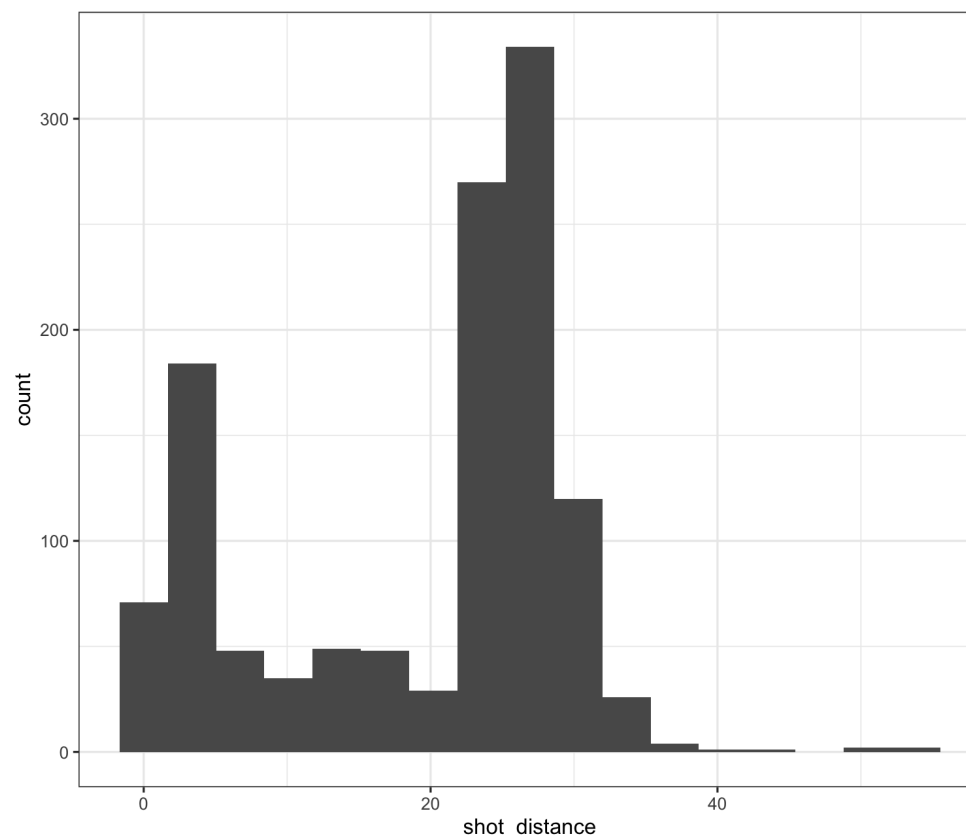
Back to histograms...

```
fd_bw <- 2 * IQR(curry_shots$shot_distance) /  
curry_shots %>%  
  ggplot(aes(x = shot_distance)) +  
  geom_histogram(binwidth = fd_bw) +  
  theme_bw()
```

- Split observed data into **bins**
- **Count** number of observations in each bin

Need to choose the number of bins, adjust with:

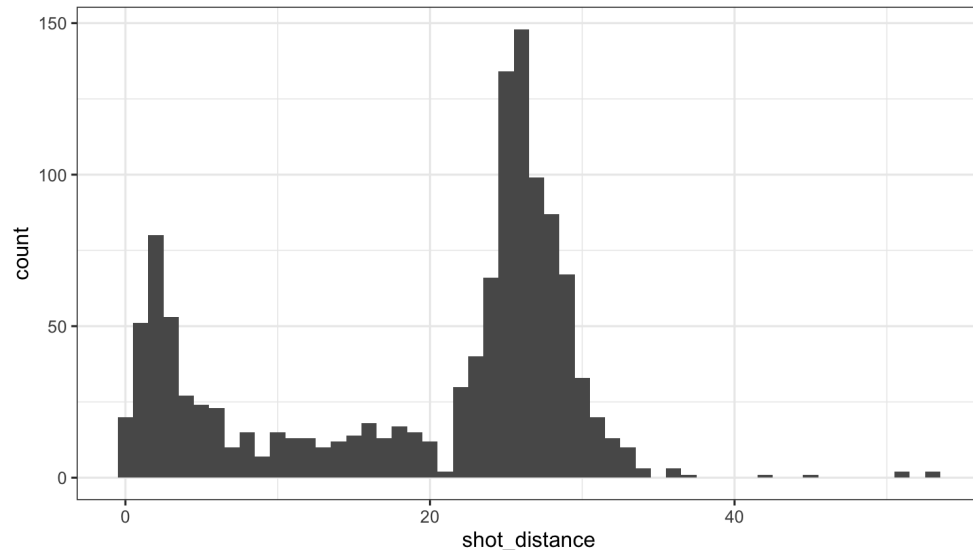
- bins - number of bins (default is 30)
- binwidth - literally the width of bins (overrides bins), various **rules of thumb**
 - e.g., see fd_bw for **Freedman–Diaconis rule**
- breaks - vector of bin boundaries (overrides both bins and binwidth)



Adjusting the bin width

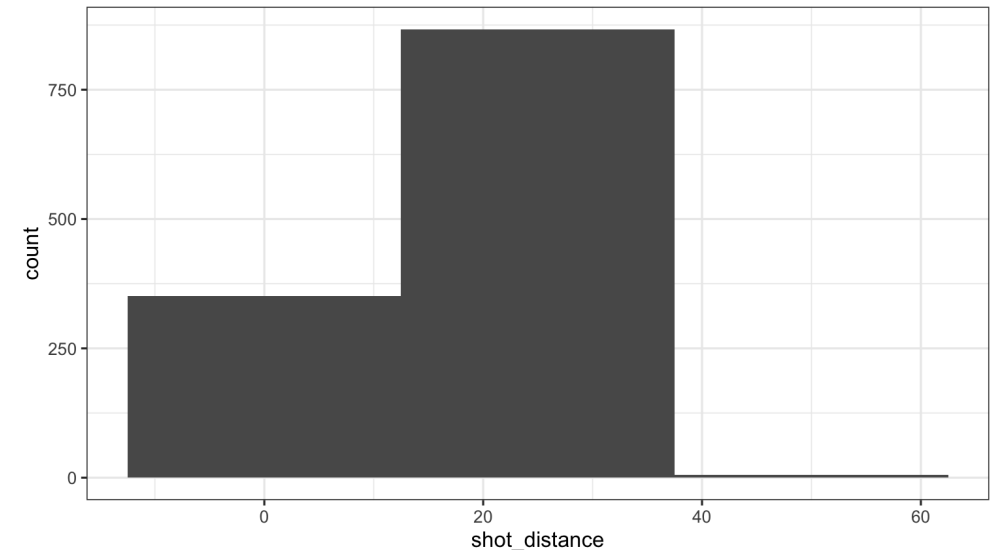
Small binwidth → *"undersmooth"* / spiky

```
curry_shots %>%  
  ggplot(aes(x = shot_distance)) +  
  geom_histogram(binwidth = 1) +  
  theme_bw()
```



Large binwidth → *"oversmooth"* / flat

```
curry_shots %>%  
  ggplot(aes(x = shot_distance)) +  
  geom_histogram(binwidth = 25) +  
  theme_bw()
```



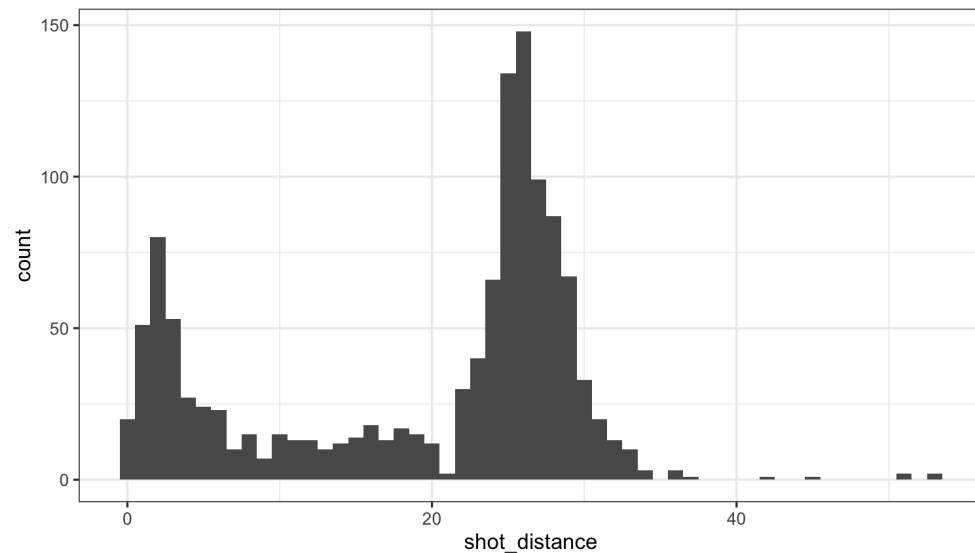
Try several approaches, the R / ggplot2 default is NOT guaranteed to be an optimal choice

A subtle point about the histogram code...

By default the bins are centered on the integers...

- left-closed, right-open intervals
- starting at -0.5 to 0.5, 0.5 to 1.5, ...

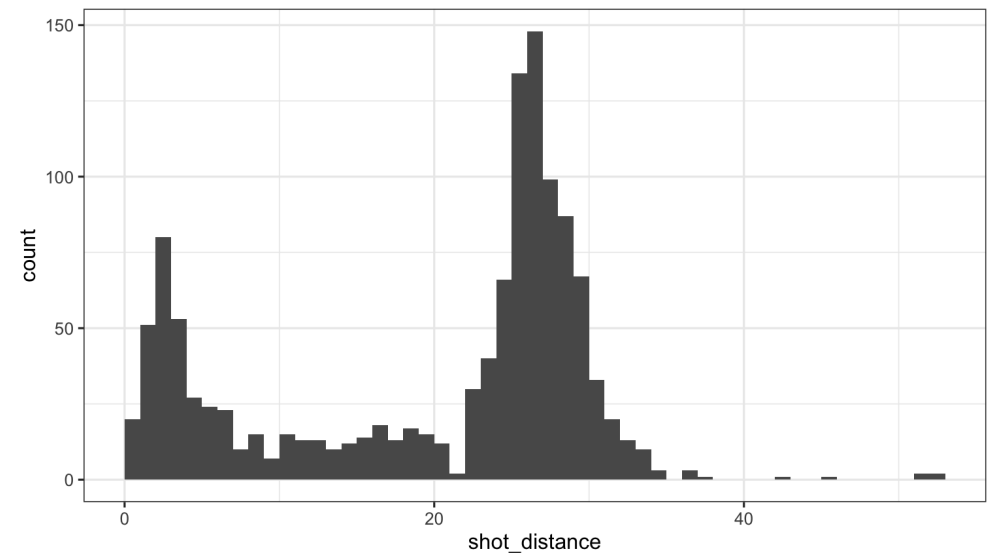
```
curry_shots %>%  
  ggplot(aes(x = shot_distance)) +  
  geom_histogram(binwidth = 1) +  
  theme_bw()
```



Specify center of one bin (e.g. 0.5)

- Reminder to use `closed = "left"`...

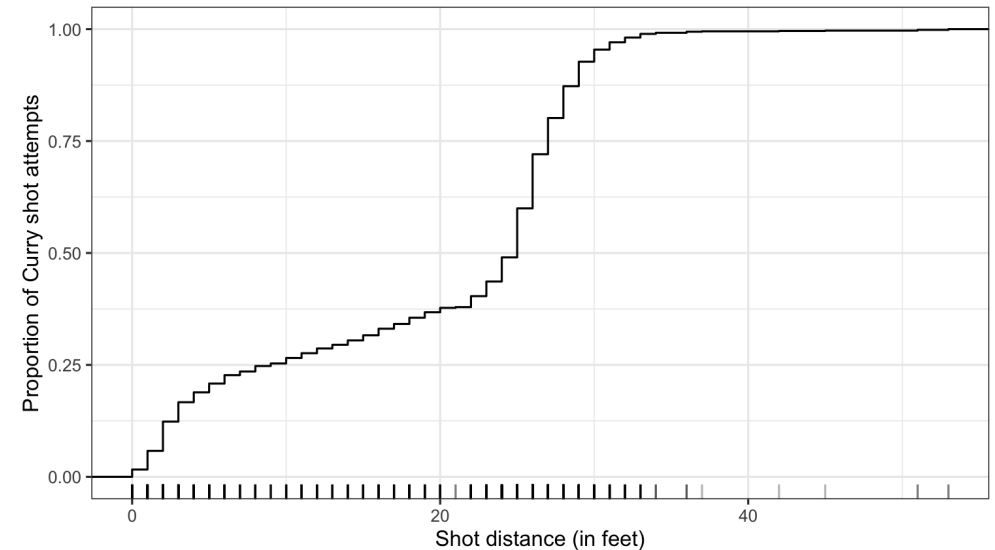
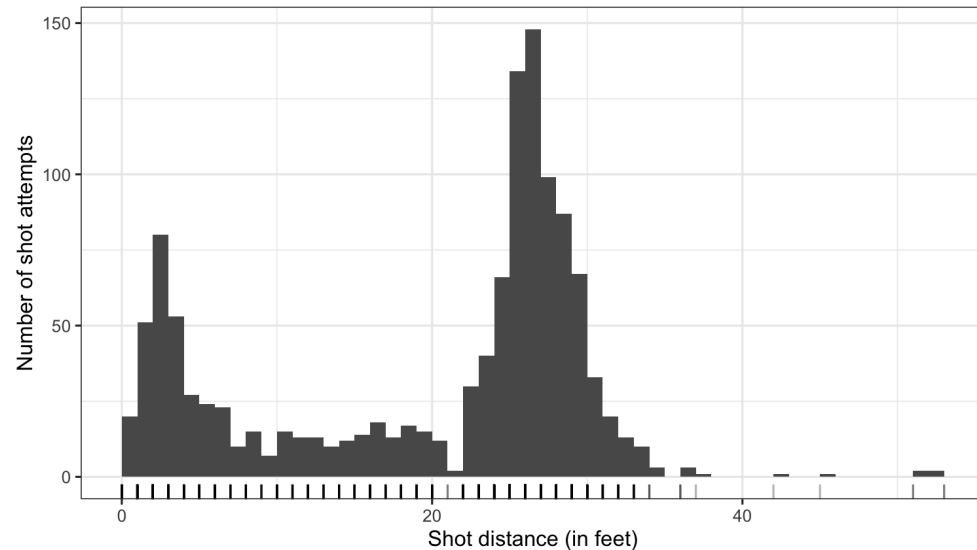
```
curry_shots %>%  
  ggplot(aes(x = shot_distance)) +  
  geom_histogram(binwidth = 1, center = 0.5,  
                 closed = "left") +  
  theme_bw()
```



How do histograms relate to the PDF and CDF?

Remember: we use the **probability density function (PDF)** to provide a **relative likelihood**

- PDF is the **derivative** of the cumulative distribution function (CDF)
- Histograms approximate the PDF with bins, and **points are equally likely within a bin**



What can say about the relative likelihood of data we have not observed?

- we want **non-zero density** between our observations, e.g., just beyond 20 feet

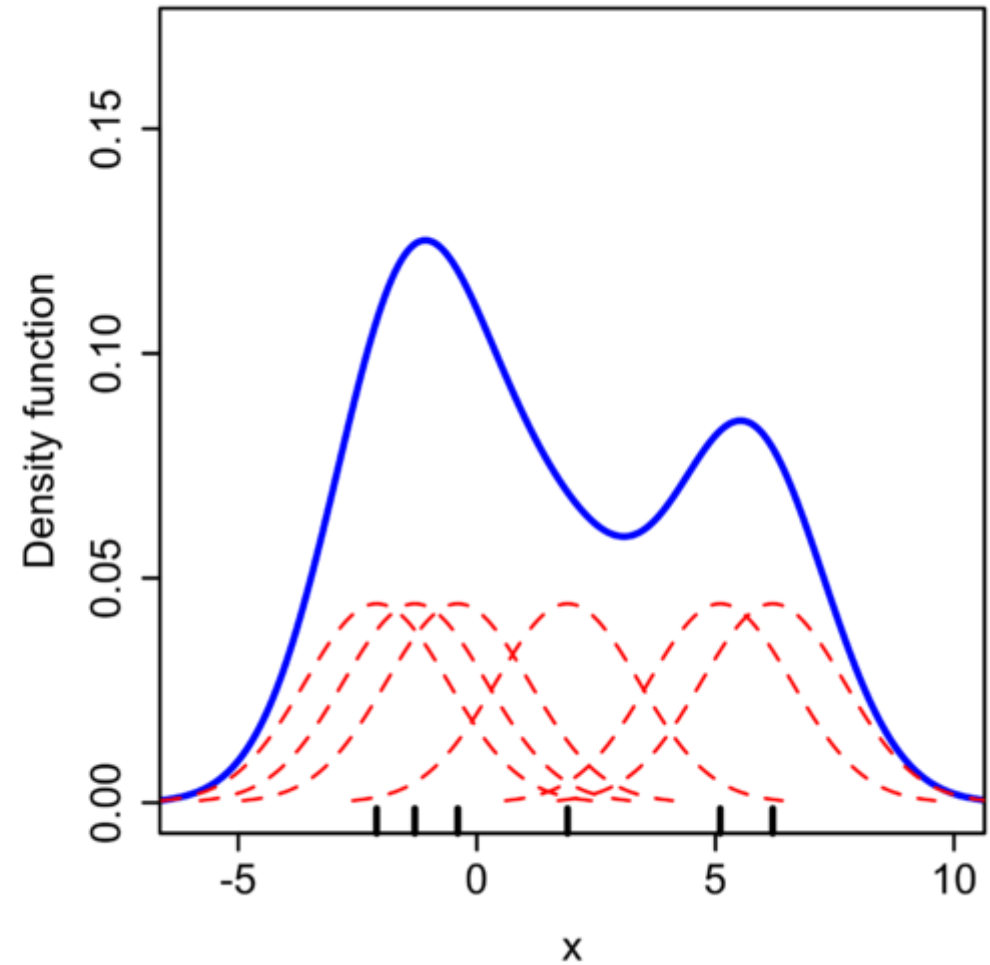
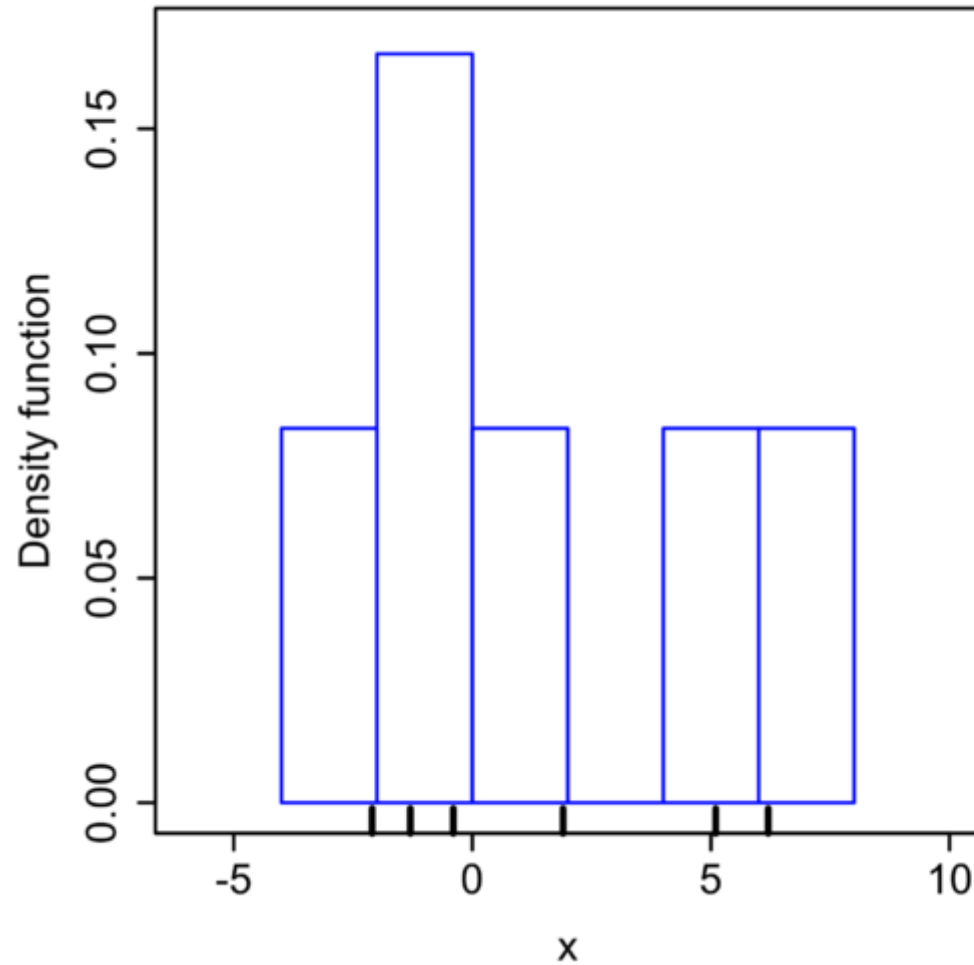
Kernel density estimation

Goal: estimate the PDF $f(x)$ for all possible values (assuming it is continuous / smooth)

$$\text{Kernel density estimate: } \hat{f}(x) = \frac{1}{n} \sum_{i=1}^n \frac{1}{h} K_h(x - x_i)$$

- n = sample size, x = new point to estimate $f(x)$ (does NOT have to be in dataset!)
- h = **bandwidth**, analogous to histogram bin width, ensures $\hat{f}(x)$ integrates to 1
- x_i = i th observation in dataset
- $K_h(x - x_i)$ is the **Kernel** function, creates **weight** given distance of i th observation from new point
 - as $|x - x_i| \rightarrow \infty$ then $K_h(x - x_i) \rightarrow 0$, i.e. further apart i th row is from x , smaller the weight
 - as **bandwidth** $h \uparrow$ weights are more evenly spread out (as $h \downarrow$ more concentrated around x)
 - typically use **Gaussian / Normal** kernel: $\propto e^{-(x-x_i)^2/2h^2}$
 - $K_h(x - x_i)$ is large when x_i is close to x

Wikipedia example

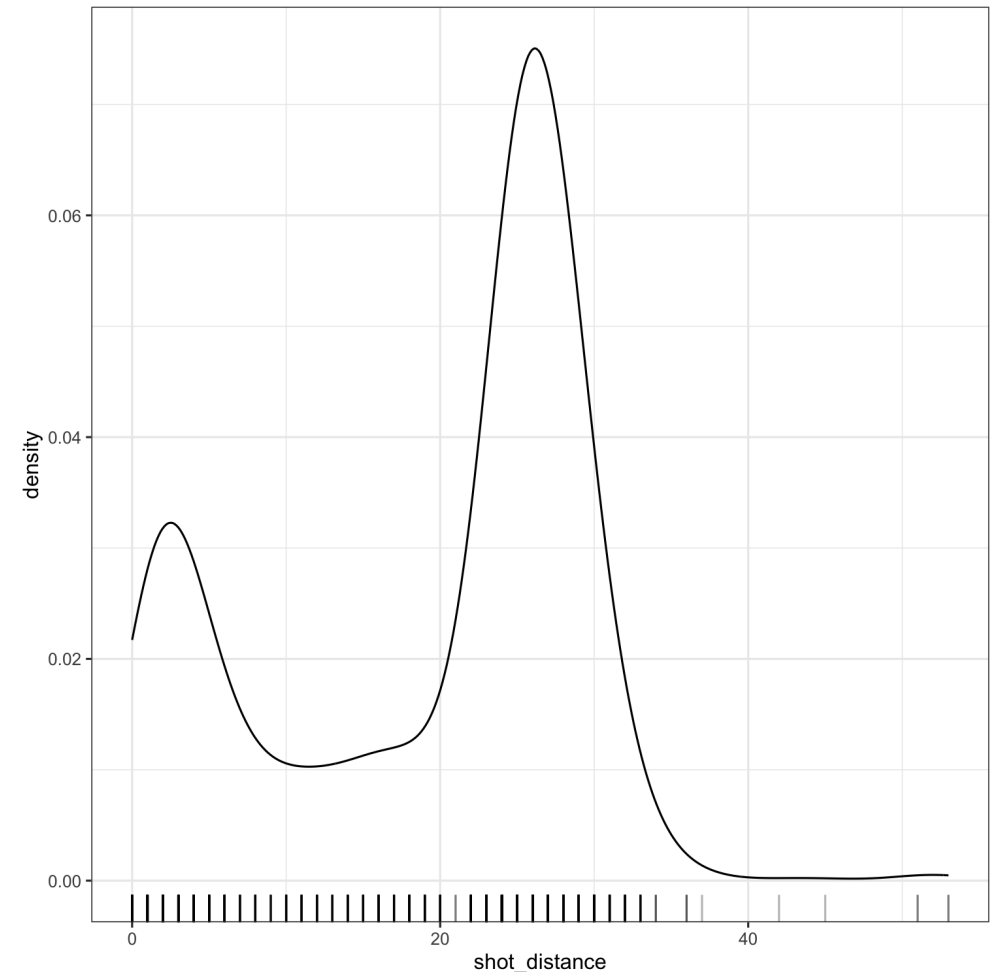


How do we compute and display the density estimate?

- We make **kernel density estimates** with `geom_density()`

```
curry_shots %>%  
  ggplot(aes(x = shot_distance)) +  
  geom_density() +  
  geom_rug(alpha = 0.3) +  
  theme_bw()
```

- **Pros:**
 - Displays full shape of distribution
 - Can easily layer
 - Add categorical variable with color
- **Cons:**
 - Need to pick bandwidth and kernel...

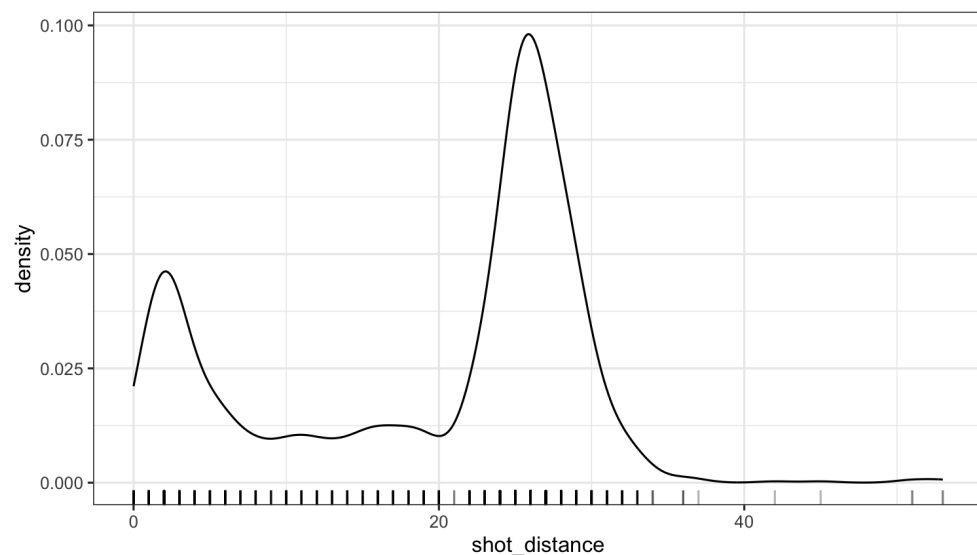


What about the bandwidth? See Chapter 14 for more...

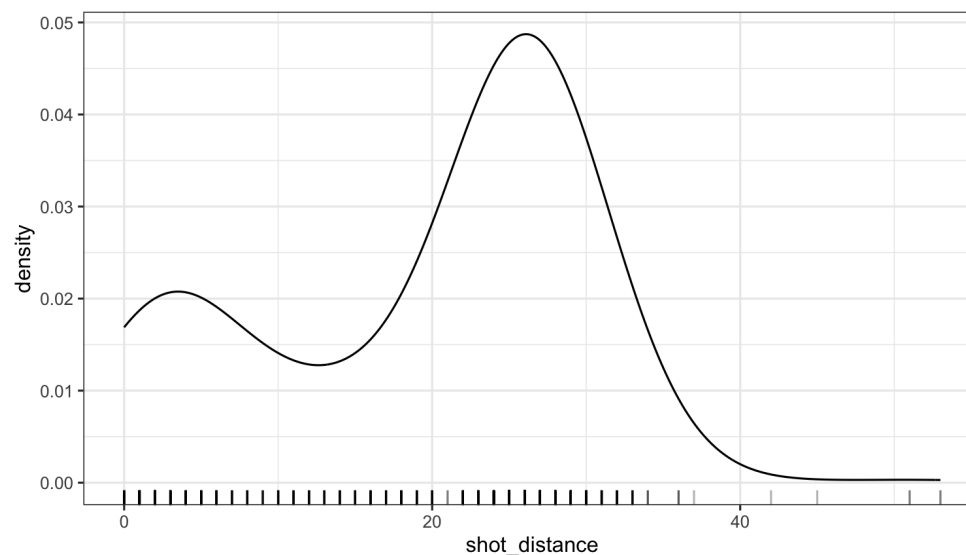
Use **Gaussian reference rule** (*rule-of-thumb*) $\approx 1.06 \cdot \sigma \cdot n^{-1/5}$, where σ is the observed standard deviation

Modify the bandwidth using the `adjust` argument - **value to multiply default bandwidth by**

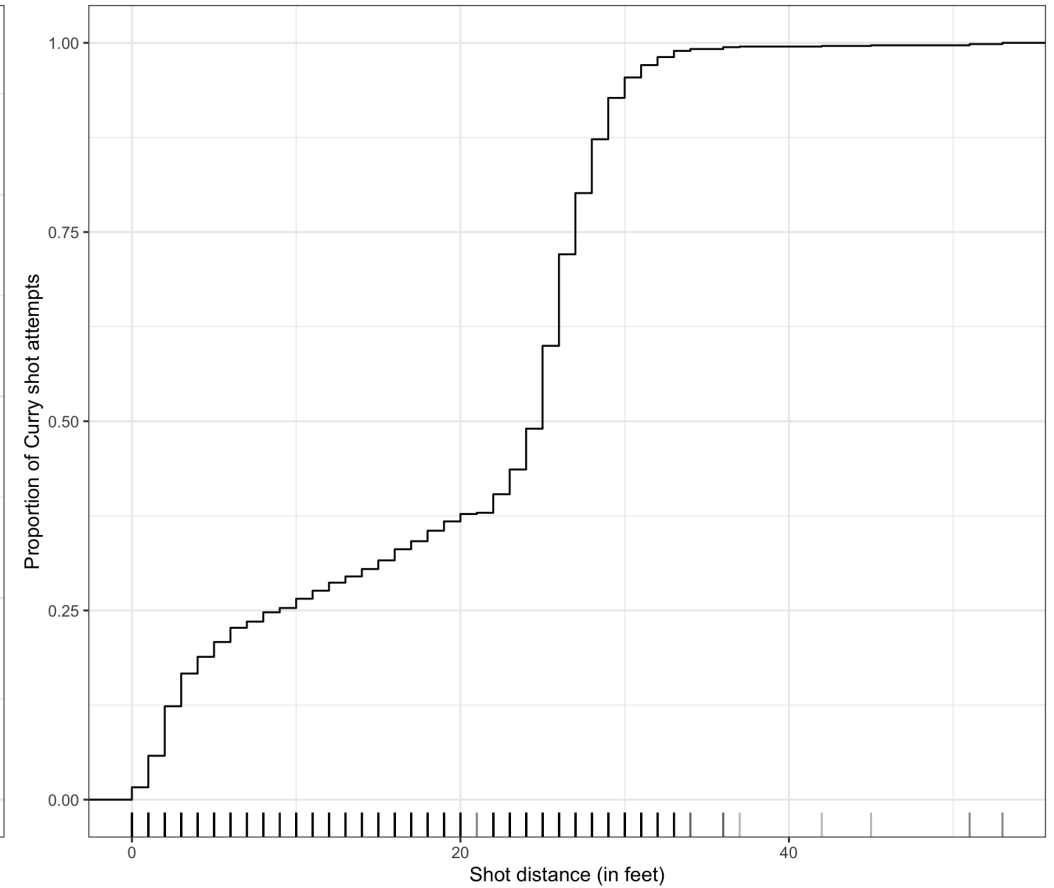
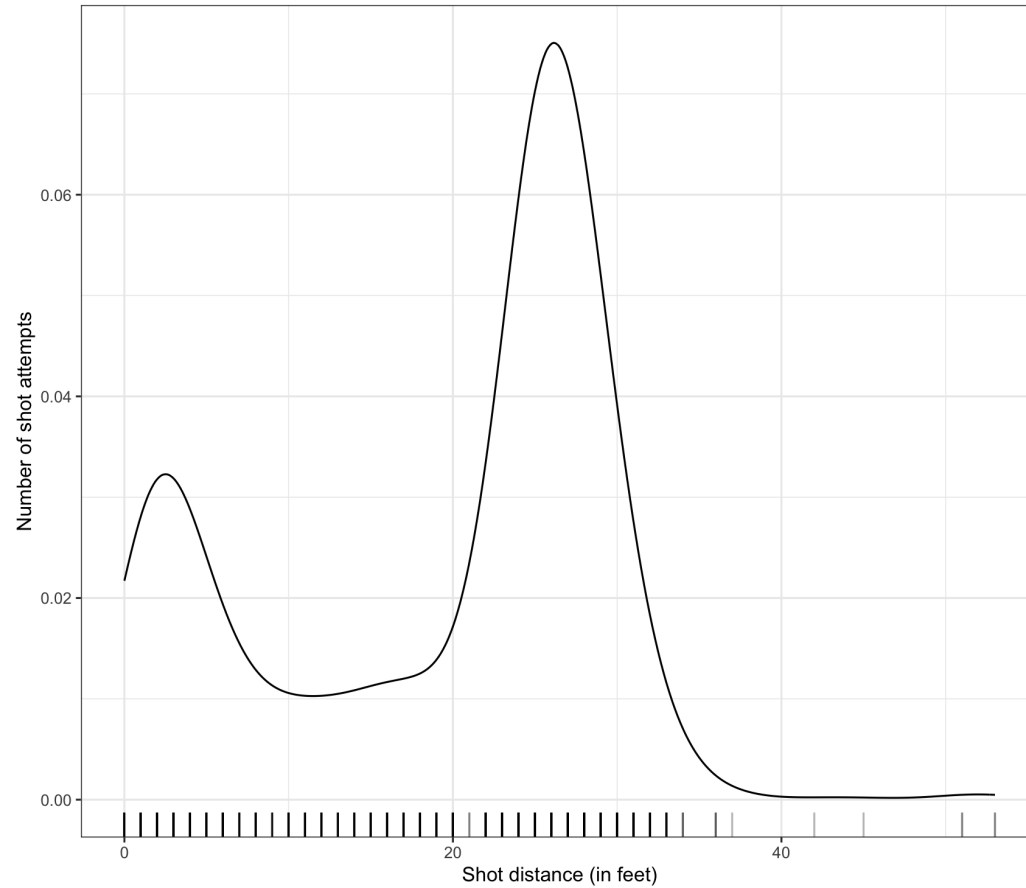
```
curry_shots %>%  
  ggplot(aes(x = shot_distance)) +  
  geom_density(adjust = 0.5) +  
  geom_rug(alpha = 0.3) + theme_bw()
```



```
curry_shots %>%  
  ggplot(aes(x = shot_distance)) +  
  geom_density(adjust = 2) +  
  geom_rug(alpha = 0.3) + theme_bw()
```



Use density curves and ECDFs together

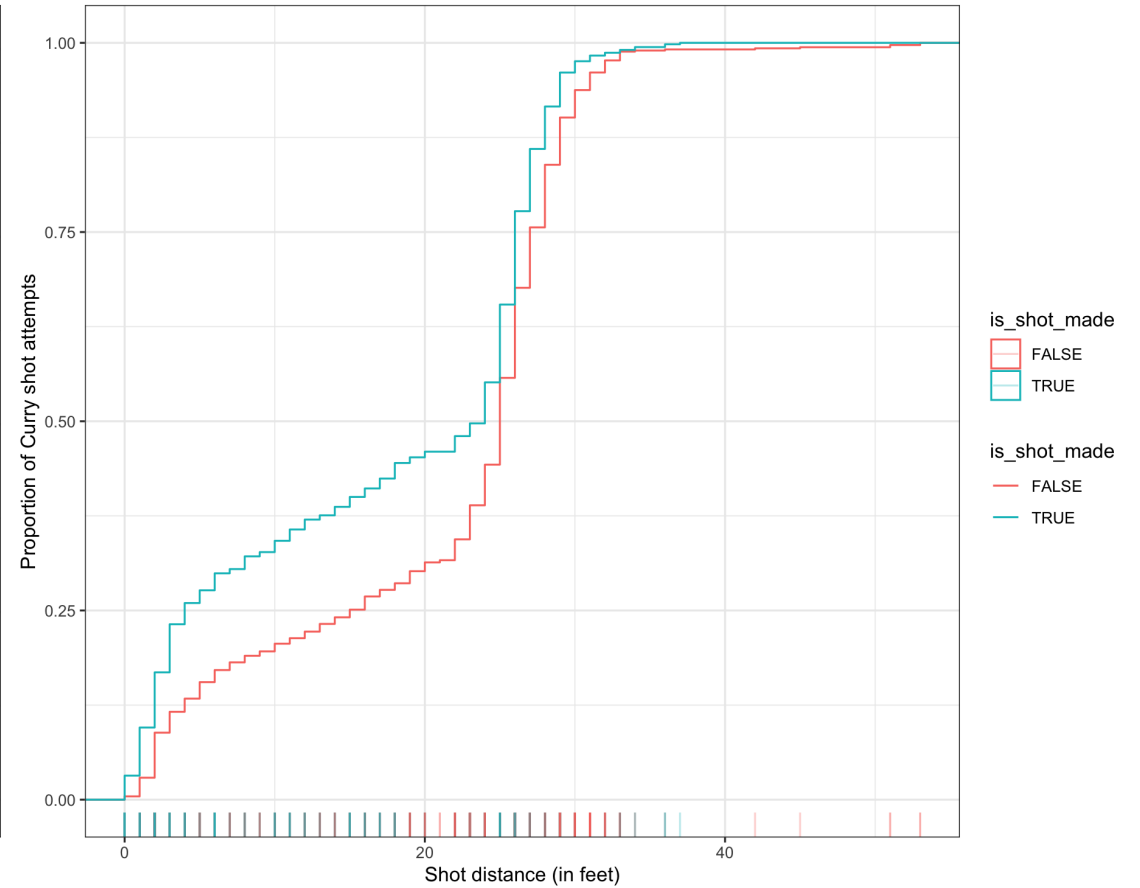
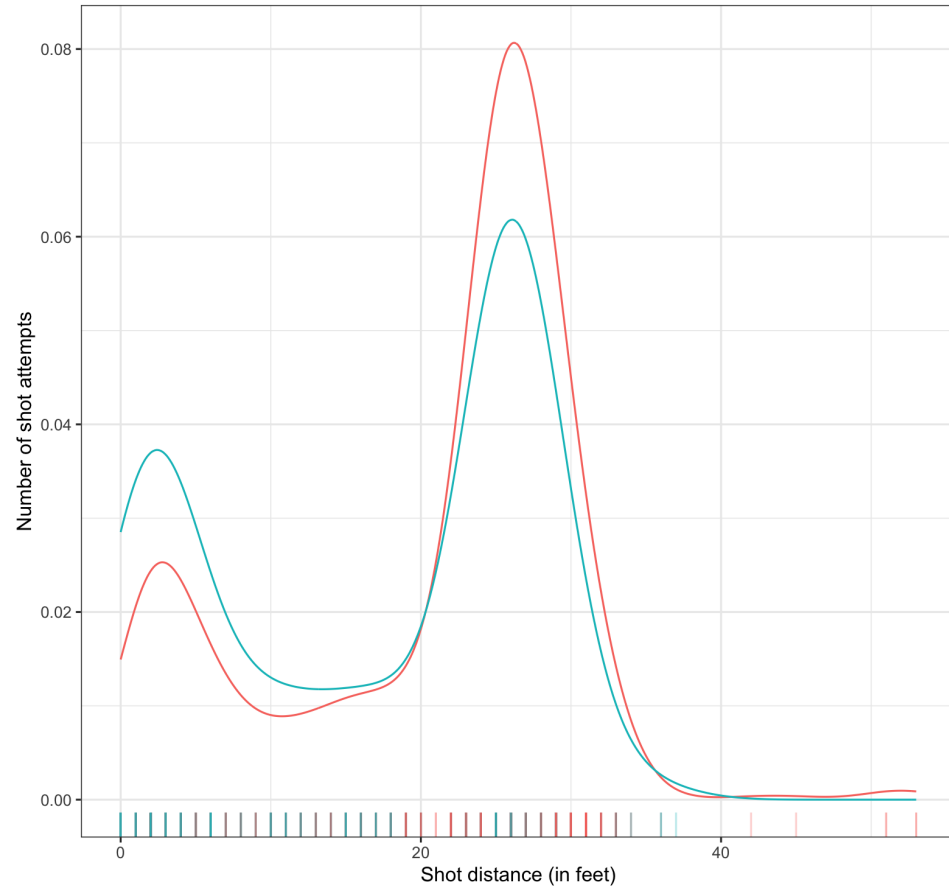


Code interlude: easy way to arrange multiple figures

Use the new `patchwork` package to easily arrange your plots (see also `cowplot`)

```
library(patchwork)
curry_shot_dens <- curry_shots %>%
  ggplot(aes(x = shot_distance)) +
  geom_density() +
  geom_rug(alpha = 0.3) +
  theme_bw() +
  labs(x = "Shot distance (in feet)",
       y = "Number of shot attempts")
curry_shot_ecdf <- curry_shots %>%
  ggplot(aes(x = shot_distance)) +
  stat_ecdf() +
  geom_rug(alpha = 0.3) +
  theme_bw() +
  labs(x = "Shot distance (in feet)",
       y = "Proportion of Curry shot attempts")
curry_shot_dens + curry_shot_ecdf
```

Use density curves and ECDFs together



Another code interlude: collect the legends

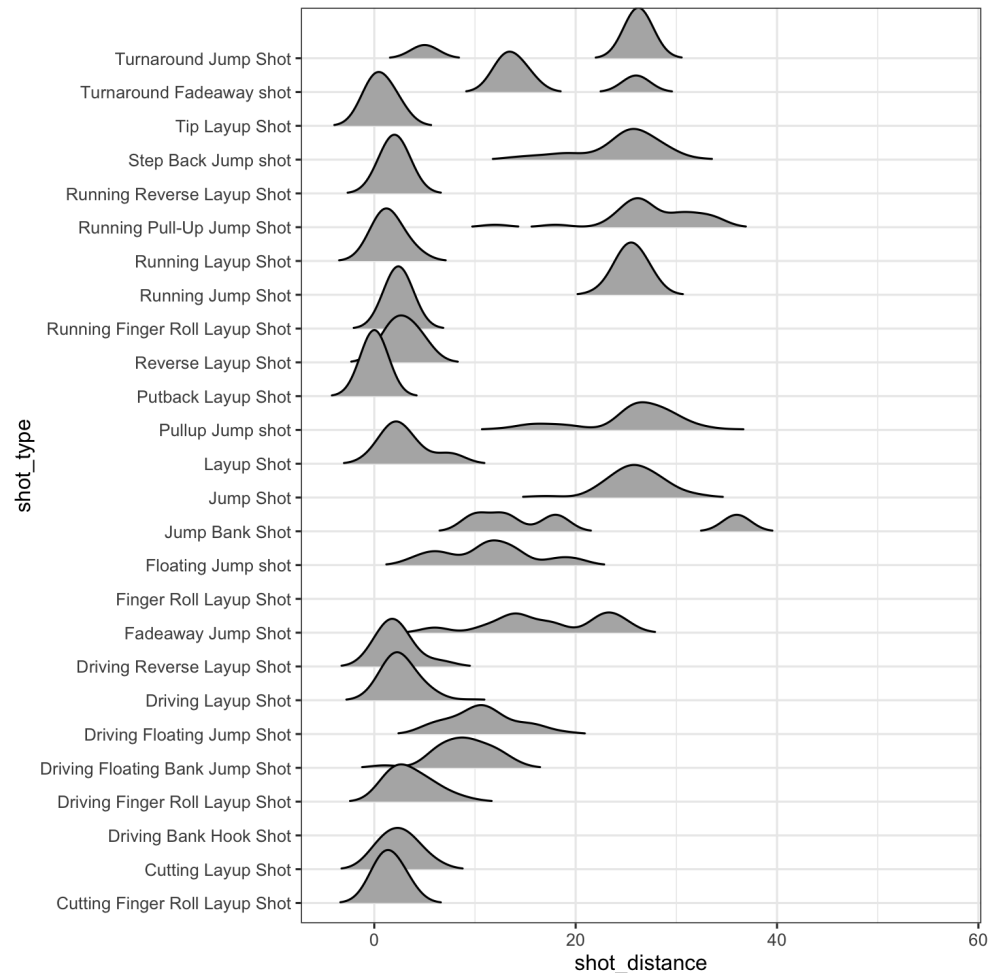
```
curry_shot_dens_made <- curry_shots %>%  
  ggplot(aes(x = shot_distance,  
             color = is_shot_made)) +  
  geom_density() +  
  geom_rug(alpha = 0.3) +  
  theme_bw() +  
  labs(x = "Shot distance (in feet)",  
        y = "Number of shot attempts")  
curry_shot_ecdf_made <- curry_shots %>%  
  ggplot(aes(x = shot_distance,  
             color = is_shot_made)) +  
  stat_ecdf() +  
  geom_rug(alpha = 0.3) +  
  theme_bw() +  
  labs(x = "Shot distance (in feet)",  
        y = "Proportion of Curry shot attempts")  
curry_shot_dens_made + curry_shot_ecdf_made + plot_layout(guides = 'collect')
```

Alternative to violins - ridge plots

- Check out the **ggribes** package for a variety of customization options

```
library(ggribes)
curry_shots %>%
  ggplot(aes(x = shot_distance,
             y = shot_type)) +
  geom_density_ridges(rel_min_height = 0.01)
  theme_bw()
```

- Useful to display conditional distributions across many levels

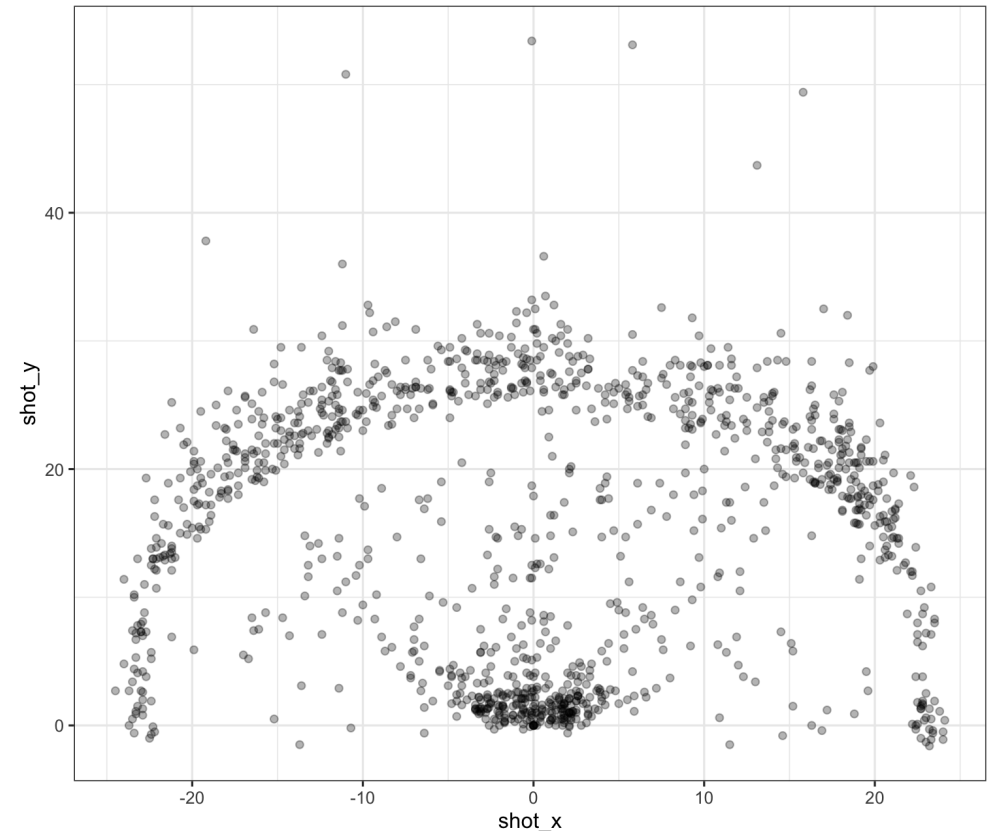


What about for 2D? (two continuous variables)

We can visualize all of the shot locations: (shot_x, shot_y)

```
curry_shots %>%  
  # Modify the shot coordinates  
  mutate(shot_x = -shot_x / 10,  
         shot_y = shot_y / 10) %>%  
  ggplot(aes(x = shot_x, y = shot_y)) +  
  geom_point(alpha = 0.3) +  
  theme_bw()
```

- Adjust transparency with alpha for overlapping points

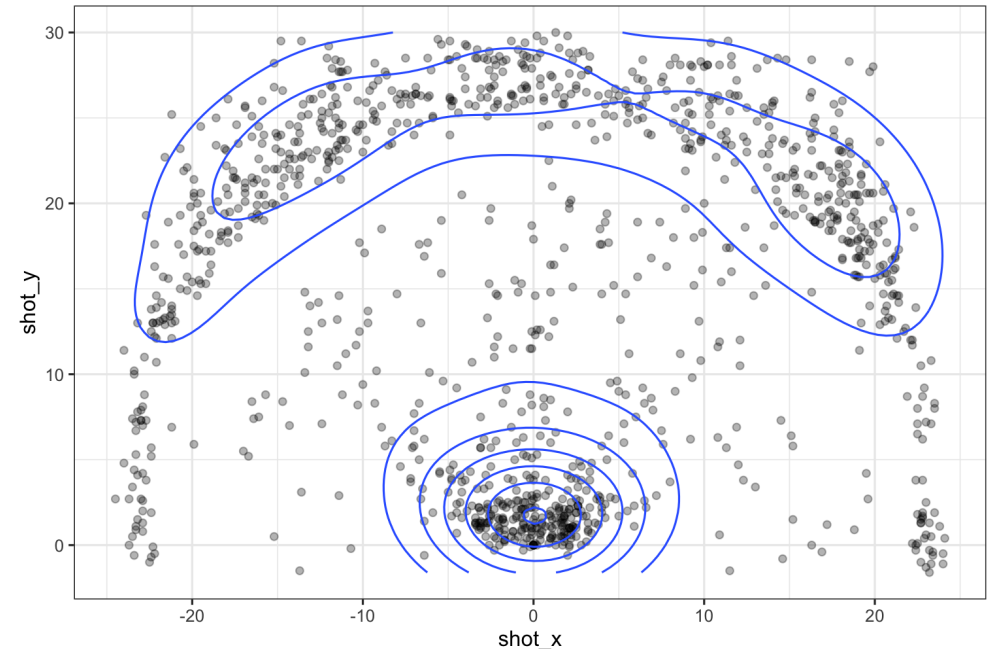


Create contours of 2D kernel density estimate (KDE)

- We make 2D KDE **contour** plots using `geom_density2d()`

```
curry_shots %>%  
  # Modify the shot coordinates  
  mutate(shot_x = -shot_x / 10,  
          shot_y = shot_y / 10) %>%  
  filter(shot_y <= 30) %>%  
  ggplot(aes(x = shot_x, y = shot_y)) +  
  geom_point(alpha = 0.3) +  
  geom_density2d() +  
  theme_bw() + theme(legend.position = "bottom")  
  coord_fixed()
```

- Extend KDE for joint density estimates in 2D (see [section 14.4.2 for details](#))
- `coord_fixed()` forced a fixed ratio

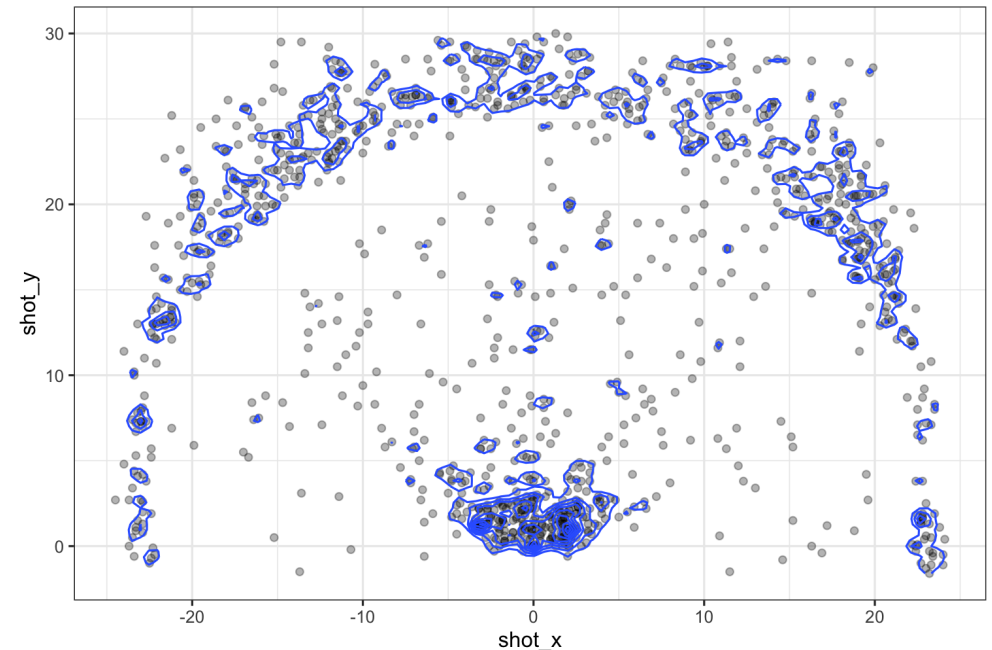


Create contours of 2D kernel density estimate (KDE)

- We make 2D KDE **contour** plots using `geom_density2d()`

```
curry_shots %>%  
  # Modify the shot coordinates  
  mutate(shot_x = -shot_x / 10,  
         shot_y = shot_y / 10) %>%  
  # Remove the outlier shots:  
  filter(shot_y <= 30) %>%  
  ggplot(aes(x = shot_x, y = shot_y)) +  
  geom_point(alpha = 0.3) +  
  geom_density2d(adjust = 0.1) +  
  theme_bw() +  
  theme(legend.position = "bottom") +  
  coord_fixed()
```

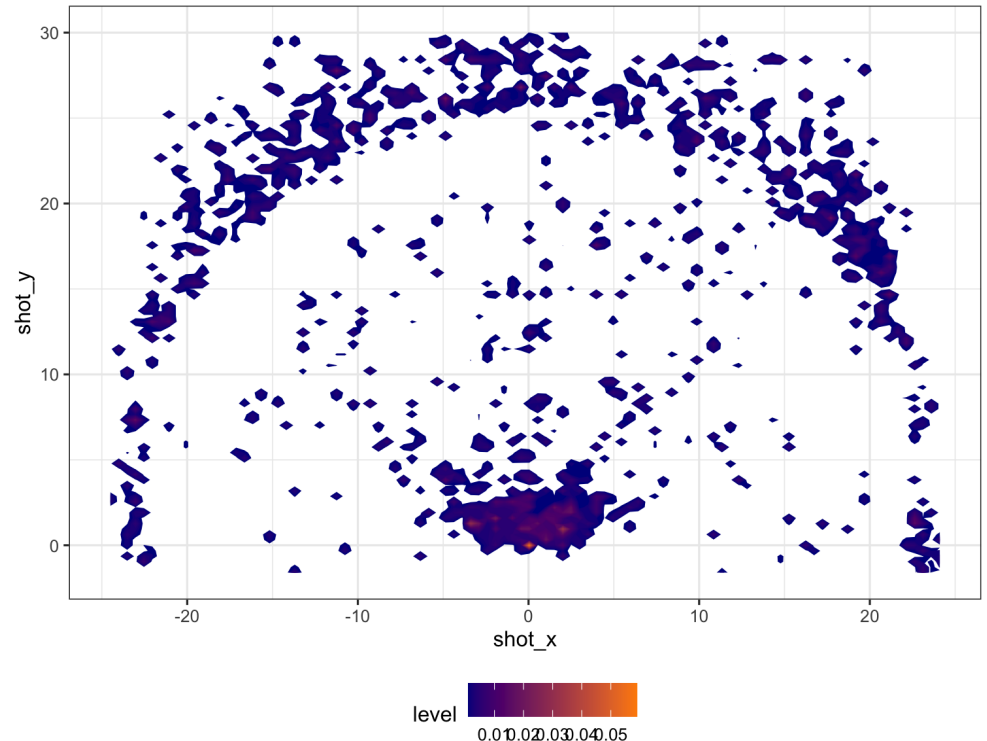
- Can use `adjust` to modify the multivariate bandwidth



Contours are difficult... let's make a heatmap instead

- We make 2D KDE **heatmap** plots using `stat_density_2d()` and the `..` or `after_stat()` function

```
curry_shots %>%  
  mutate(shot_x = -shot_x / 10,  
         shot_y = shot_y / 10) %>%  
  filter(shot_y <= 30) %>%  
  ggplot(aes(x = shot_x, y = shot_y)) +  
  stat_density2d(h = 0.5, bins = 60,  
               aes(fill = after_stat(level)  
                 geom = "polygon")) +  
  scale_fill_gradient(low = "darkblue",  
                    high = "darkorange") +  
  theme_bw() + theme(legend.position = "botto  
coord_fixed()
```

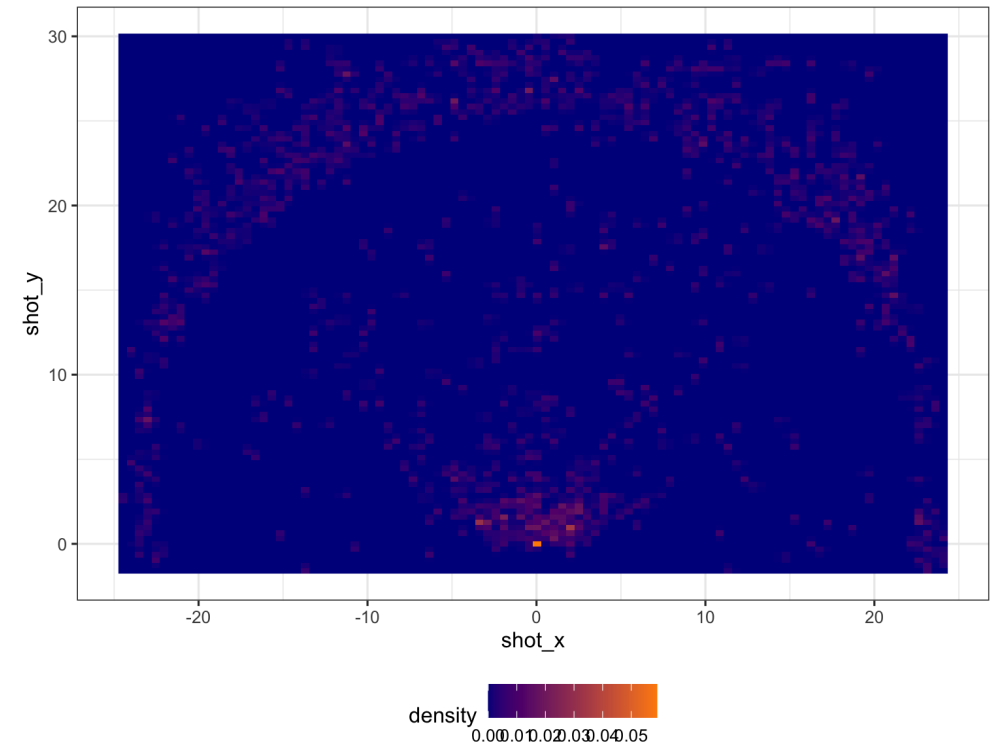


Multivariate density estimation can be difficult

Turn off contours and use tiles instead

- We make 2D KDE **heatmap** plots using `stat_density_2d()` and the `..` or `after_stat()` function

```
curry_shots %>%  
  mutate(shot_x = -shot_x / 10,  
         shot_y = shot_y / 10) %>%  
  filter(shot_y <= 30) %>%  
  ggplot(aes(x = shot_x, y = shot_y)) +  
  stat_density2d(h = 0.5, bins = 60,  
                contour = FALSE,  
                aes(fill = after_stat(density)),  
                geom = "raster") +  
  scale_fill_gradient(low = "darkblue",  
                    high = "darkorange") +  
  theme_bw() + theme(legend.position = "bottom")  
  coord_fixed()
```

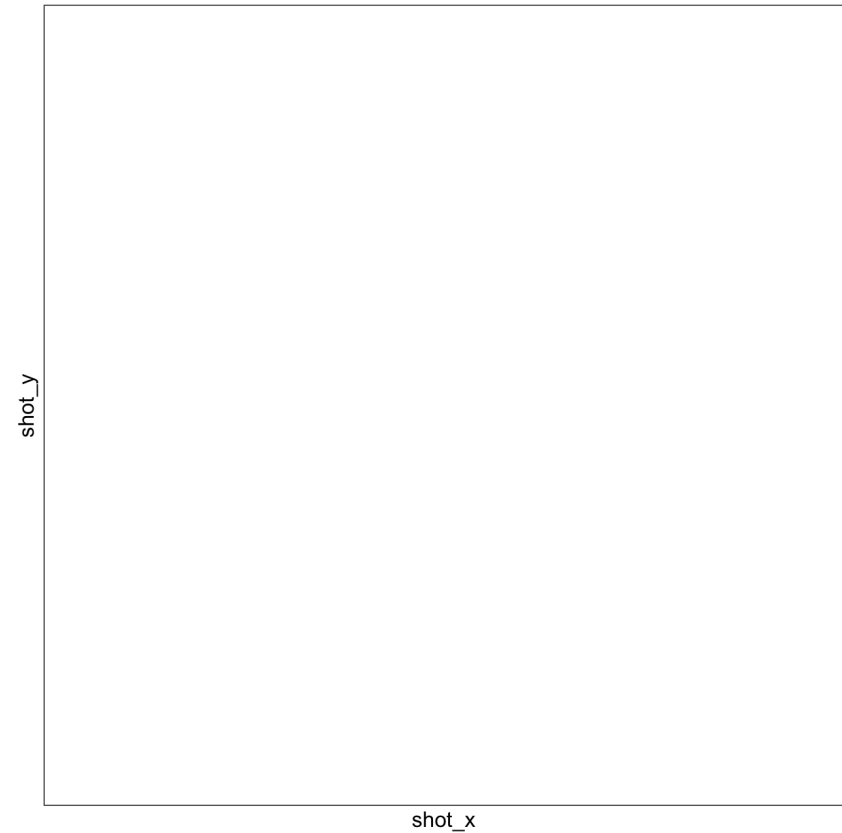


Best alternative? Hexagonal binning

- We make **hexagonal heatmap** plots using `geom_hex()`
- Need to have the `hexbin` package installed

```
curry_shots %>%  
  mutate(shot_x = -shot_x / 10,  
         shot_y = shot_y / 10) %>%  
  filter(shot_y <= 30) %>%  
  ggplot(aes(x = shot_x, y = shot_y)) +  
  geom_hex(binwidth = c(1, 1)) +  
  scale_fill_gradient(low = "darkblue",  
                     high = "darkorange") +  
  theme_bw() + theme(legend.position = "bottom")  
  coord_fixed()
```

- Can specify binwidth in both directions
- Avoids limitations from smoothing



What about his shooting efficiency?

- Can compute a function of another variable inside hexagons with `stat_summary_hex()`
- Check out [BallR](#) for code examples to make shot charts and drawing courts

```
curry_shots %>%  
  mutate(shot_x = -shot_x / 10,  
         shot_y = shot_y / 10) %>%  
  filter(shot_y <= 30) %>%  
  ggplot(aes(x = shot_x, y = shot_y,  
            z = is_shot_made,  
            group = -1)) +  
    stat_summary_hex(binwidth = c(2, 2),  
                    color = "black",  
                    fun = mean) +  
    scale_fill_gradient(low = "darkblue",  
                      high = "darkorange") +  
    theme_bw() + theme(legend.position = "bottom")  
    coord_fixed()
```

