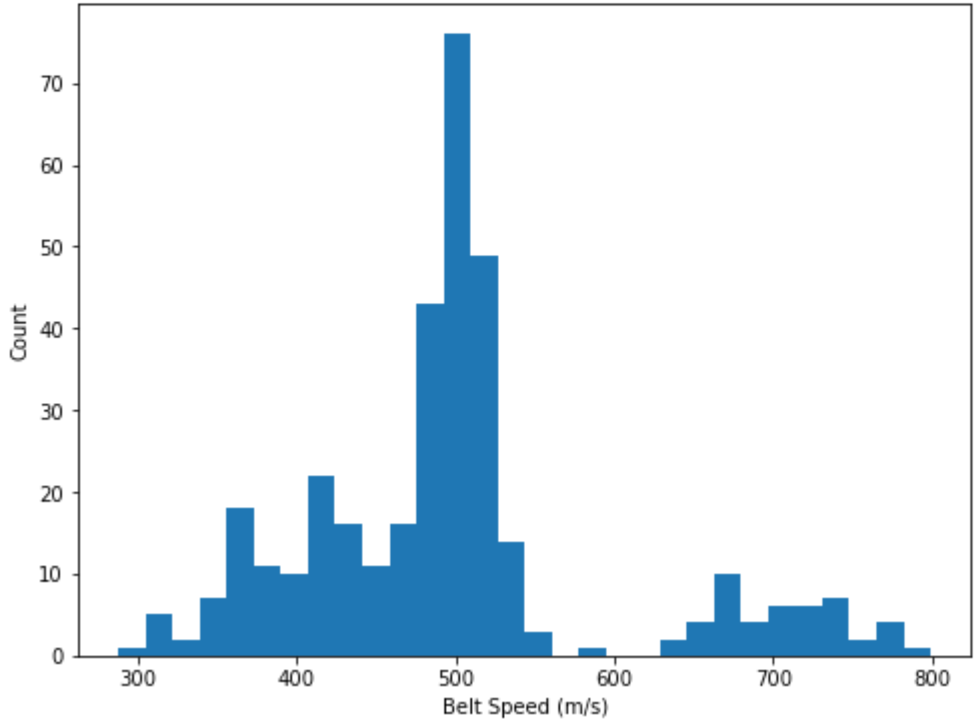
Lecture 6

Non-Parametric Density Estimation

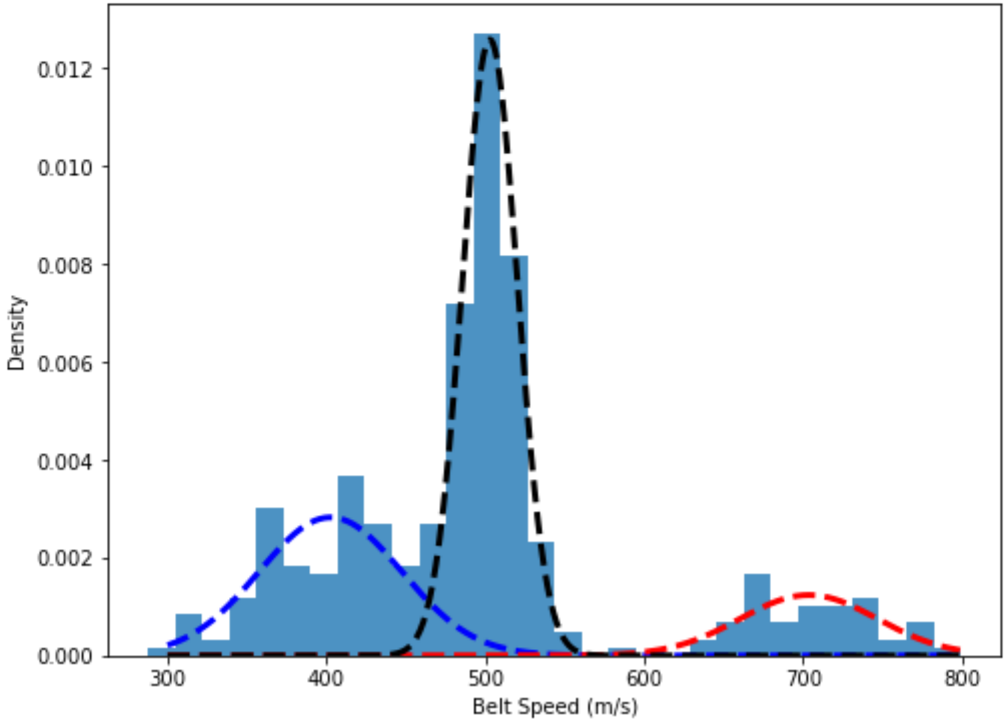
Motivating Example

We have some highly irregular data, how can we fit a distribution?



Motivating Example

What if instead of one distribution, we fit several overlapping distributions?



Mixture Models

- A **Gaussian Mixture Model** assumes the data is generated by K distributions, each with it's own mean (μ_k), covariance (Σ_k), and weight (π_k)
  - Non-Gaussian mixture models also exist, but we won't be covering those in this lecture

- To fit a Gaussian Mixture Model, we must do two things
  - Assign a probability of each data point being in each distribution
  - Fit each distribution using the methods we learned earlier

Fitting GMMs

- The likelihood given by a Gaussian Mixture Model takes the equation

$$P(X_i) = \sum_{k=1}^{K} \pi_k P(X_i | Z_i = k)$$

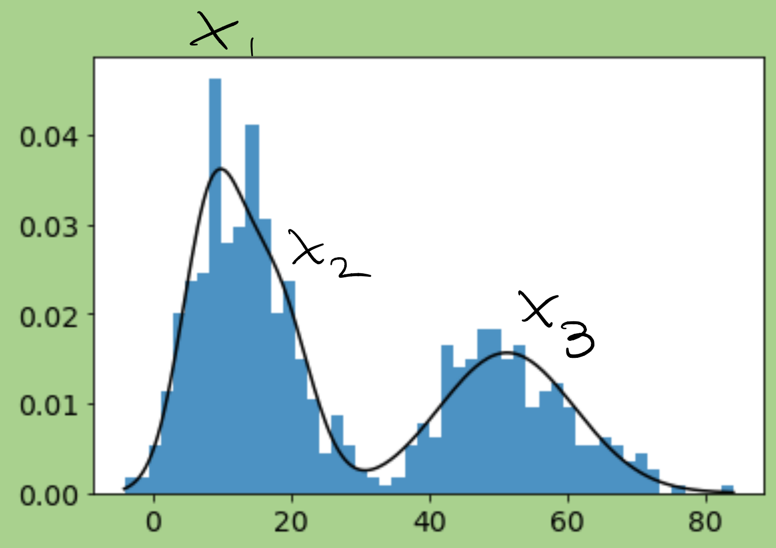
- We can optimize this using the **expectation-maximization** algorithm
  - **Expectation:** Calculate the probability of a point being in each class
  - **Maximization:** Given these estimates, calculate using weighted maximum likelihood the parameters of each distribution.

A more in depth explanation is available at https://stephens999.github.io/fiveMinuteStats/intro_to_em.html

Mixture Model Demo

```
X1 = np.random.normal(10,5,size=300)
X2 = np.random.normal(20,5,size=100)
X3 = np.random.normal(50,10,size=250)
X = np.concatenate([X1,X2,X3]).reshape(-1,1)

from sklearn.mixture import GaussianMixture
gm = GaussianMixture(n_components=3)
gm.fit(X)

xr = np.linspace(X.min(), X.max(), 200).reshape(-1,1)
density = np.exp(gm.score_samples(xr))

print(gm.means_)
print(gm.covariances_)
print(gm.weights_)
```

↳ log-likelihood



Non-Gaussian Mixture Models

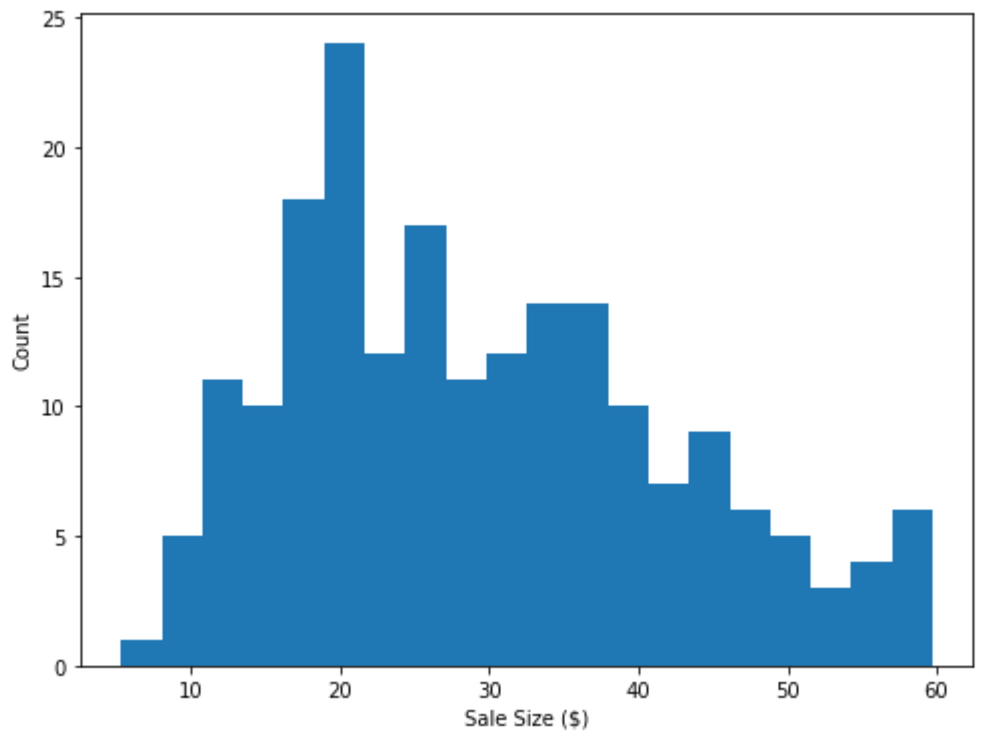
- Non-Gaussian Mixture Models do exist (*e.g.* Poisson Mixture Models) but aren’t commonly found in software packages

- You can fit any distribution mixture you want using the E-M algorithm (but I won’t ask you to in this course)

Another Example

I want to understand the distribution of how much customers spend in one purchase.

I have recorded the sale value for all my sales in the past week



Why not use parameterized models?

- **Parameterized** models estimate based on a **finite** number of parameters
  - A normal distribution is parameterized by mean (μ) and standard deviation (σ)

- Parameterized models are also limited because the real world does not follow their assumptions

- **Non-parameterized** have huge, effectively **infinite**, numbers of parameters that allow them to be more expressive

Estimating Density from Histograms

- We can directly convert a histogram into a probability density functions using the formula

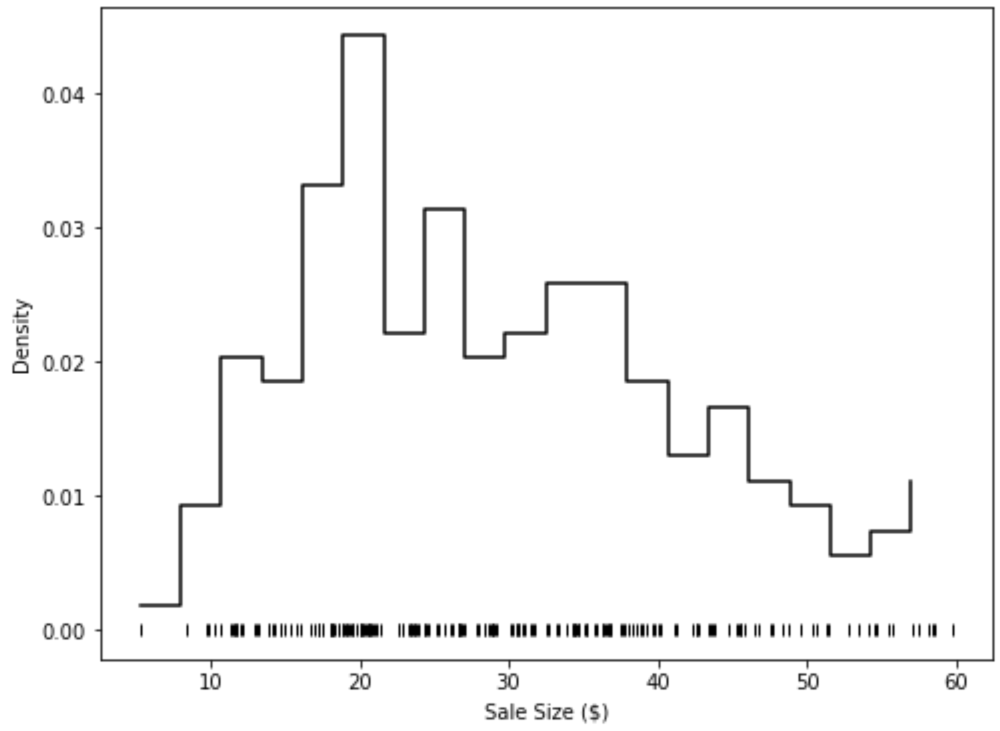
\# obs

observation

$$f_H(x) = \frac{1}{nh}\sum_{i=1}^{n} 1(X_i \textit{ in the same bin as } x)$$

bin width

point of interest



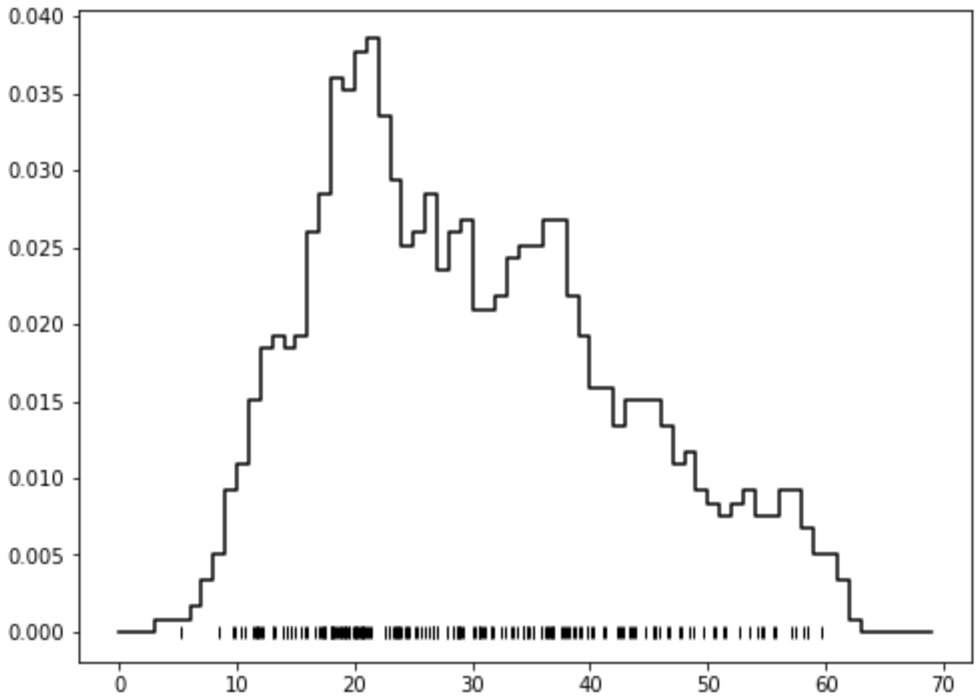
Estimating Density from Histograms

An alternative is the **Naïve Density Estimator**

This essentially sets a histogram bin centered on x

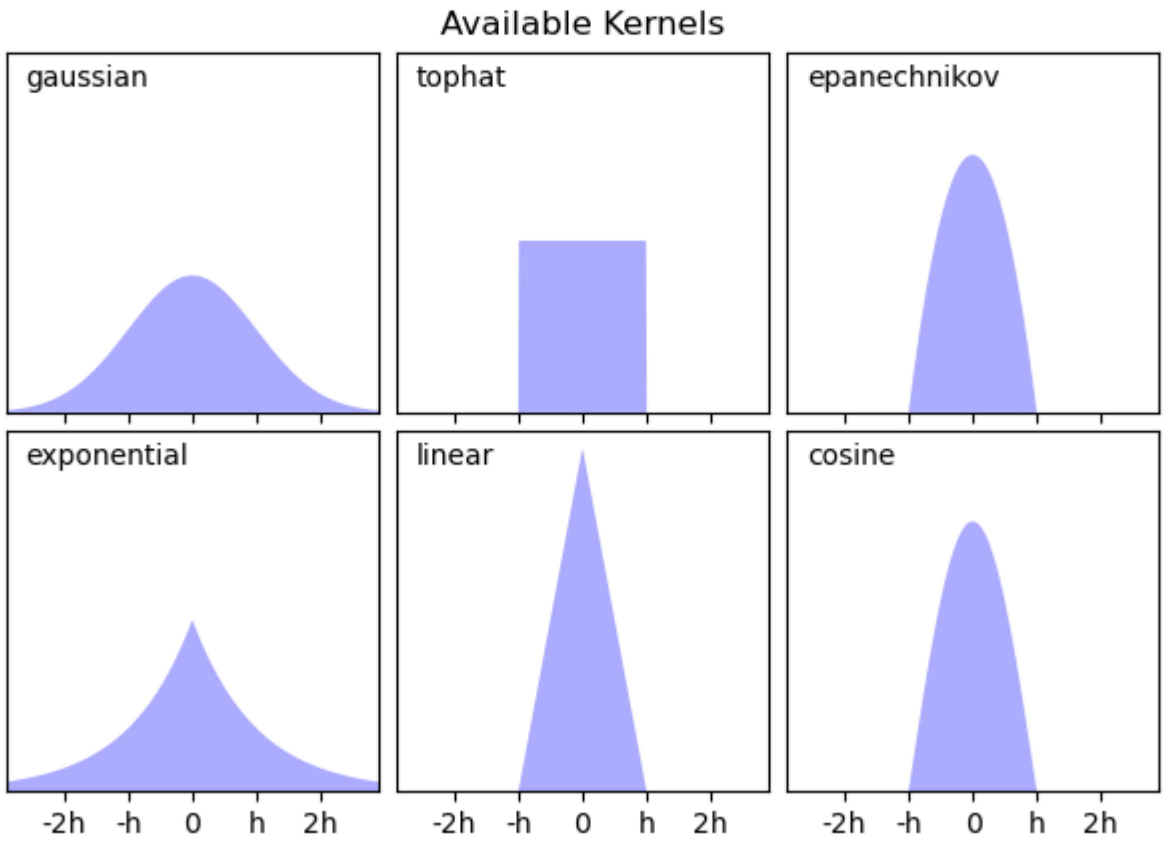
PoI   obs   bandwidth

$$f_N(x) = \frac{1}{nh}\sum_{i=1}^{n}\frac{1\big((x-h) \le x_i \le (x+h)\big)}{2}$$



Introducing Kernels

- A **kernel** is a function that weights each data point based on its distance to the point of evaluation



Estimating Density using Kernels

- Each kernel produces a probability distribution based on its data point

$$\hat{f}(x) = \frac{1}{\text{nh}} \sum_{i=1}^{n} K(z)$$

$$z = \frac{x - X_i}{h}$$

x = Point of Interest

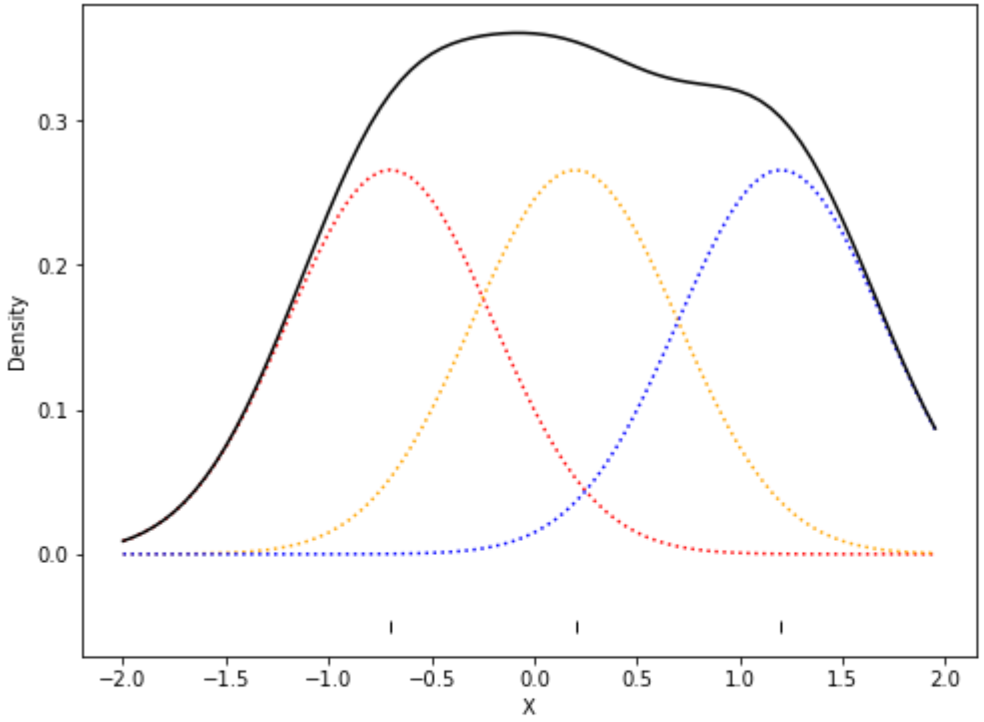
X_i = Data point / observation

h = bandwidth

Definition of non-parametric methods

- Let’s look at an example using a **Gaussian kernel**

$$\hat{f}(x) = \frac{1}{\text{nh}}\sum_{i=1}^{n}\frac{1}{\sqrt{2\pi}}e^{\frac{1}{2}(\frac{x-X_i}{h})}$$



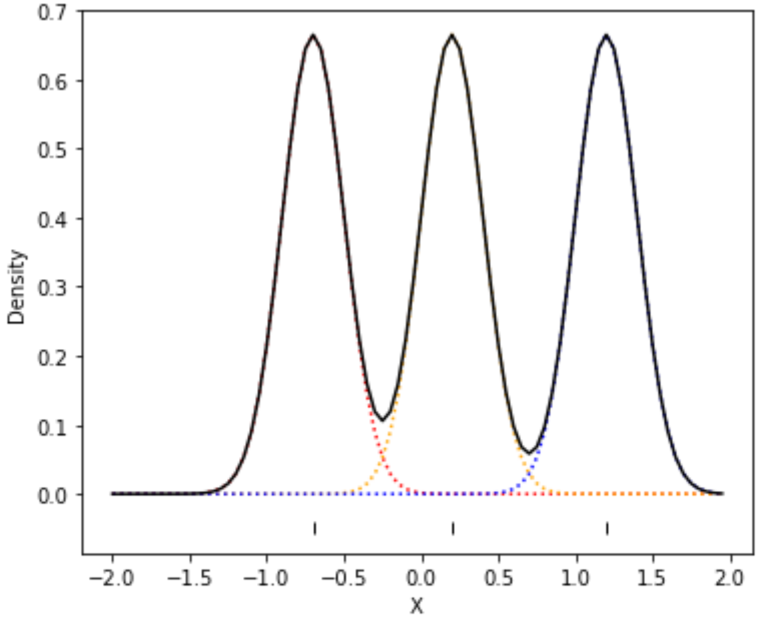
$\rightarrow K(z)$

$\rightarrow \hat{f}(x)$

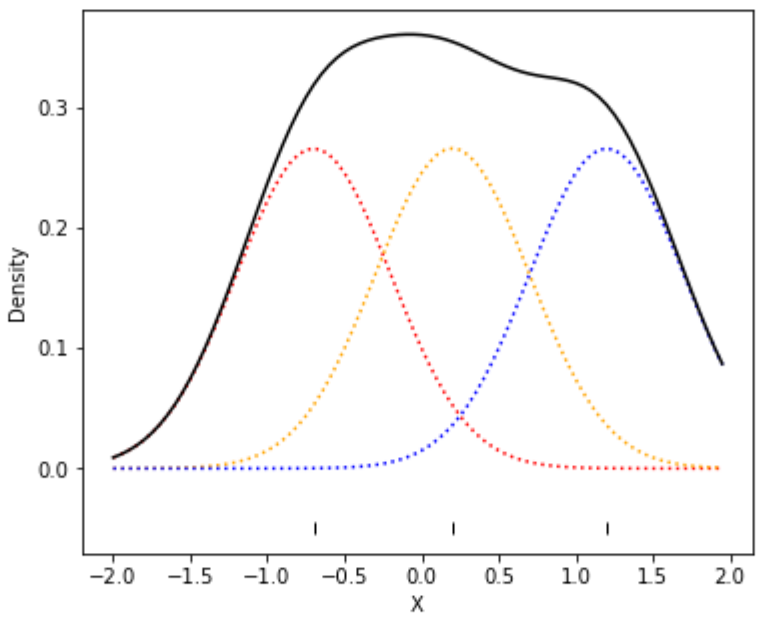
Effect of bandwidth

- **Bandwidth** is a scaling factor for the width of the kernel distributions

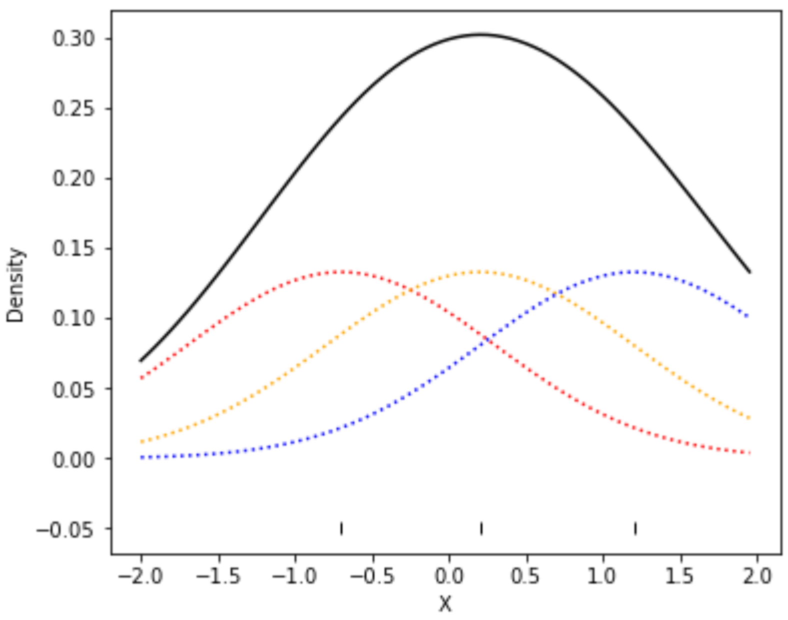
H = 0.2



H = 0.5



H = 1.0



Bandwidth selection

- The optimal bandwidth (*i.e.* bandwidth that minimizes the mean squared error) depends on the true probability distribution – which you don't know

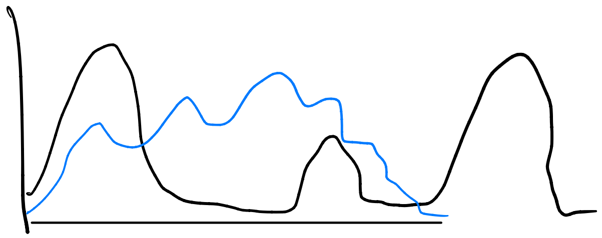
- We can approximate this for Gaussian-esque data using **Silverman's rule of thumb**

std dev estimate

bandwidth

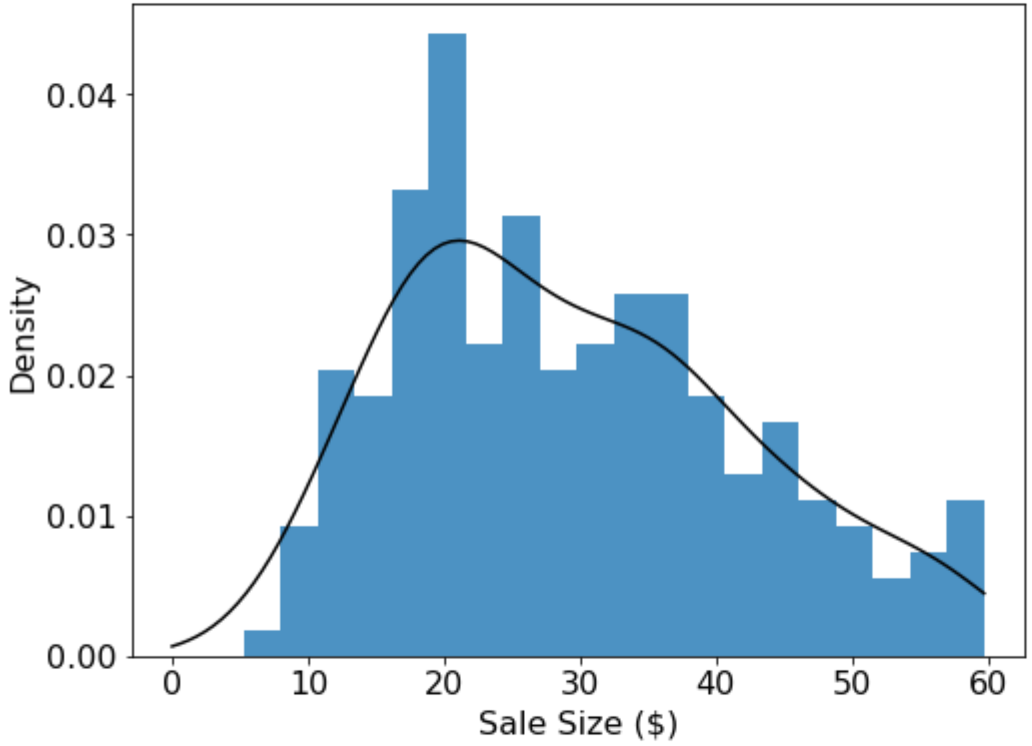
$$\hat{h} = \left(\frac{4 * \hat{\sigma}^5}{3n}\right)^{\frac{1}{5}} \approx 1.06 * \hat{\sigma} * n^{-\frac{1}{5}}$$

n: # obs



Finishing the example

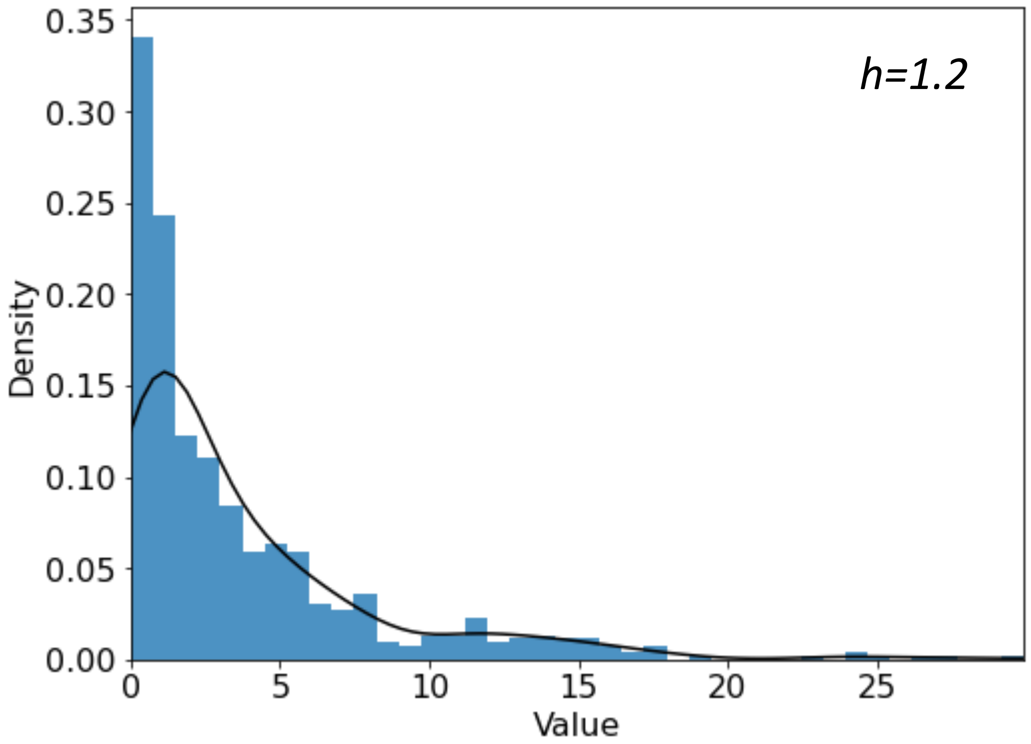
```
from sklearn.neighbors import KernelDensity

sigma_hat = np.std(df['Sales'])
n = len(df['Sales'])
h_hat = 1.06 * sigma_hat * n**-0.2
kde = KernelDensity(kernel='gaussian', bandwidth=h_hat)
        .fit(df['Sales'])

xr = np.linspace(0, df['Sales'].max(), num=100)

# KerndelDensity.score_samples returns log-likelihood,
not density.
p = np.exp(kde.score_samples(xr))
```



KernelDensity(bandwidth="silverman")

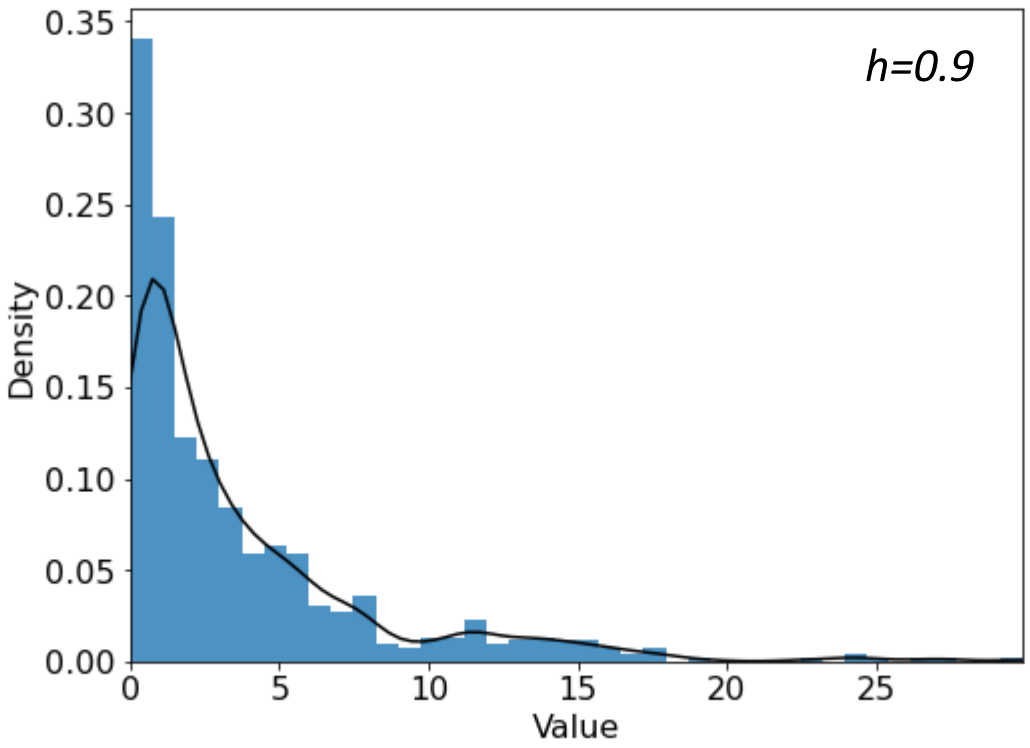
Limits of Silverman’s Rule

- Silverman’s rule of thumb assumes the data is near-normal. If your data differs from this, it tends to over-smooth



Cross-Validation Bandwidth Selection

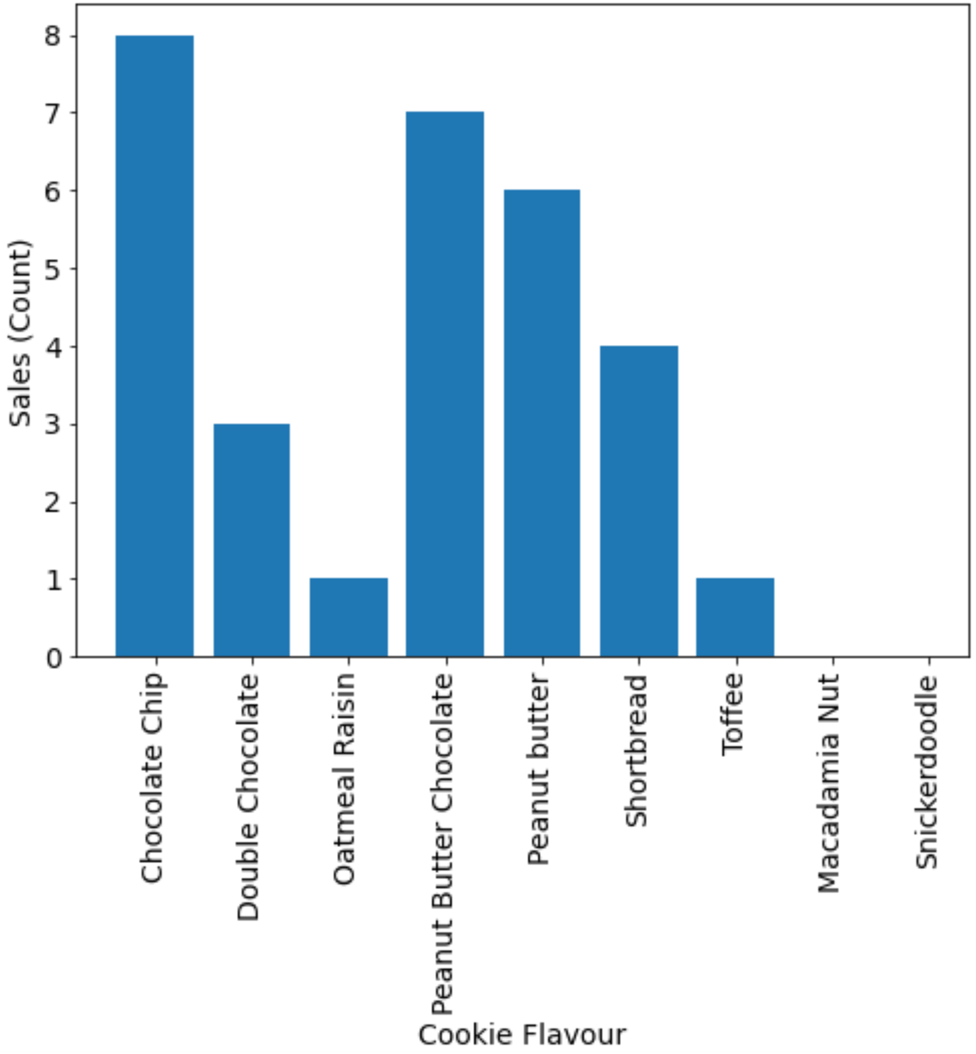
- Use grid search or another parameter optimization tool
  - Scikit-learn.model_selection has good options here



Another Example

Extending my previous study, I want to examine how many of each cookie flavour I sell.

I have recorded the flavour of the last 30 cookies sold.



Density Estimation of Categorical Variables

- The simplest method is the relative frequency estimation

$$p(x) = \frac{1}{N}\sum_{i=1}^{N} X_i = x$$

- In some cases, we may want to smooth this estimate using a smoothing kernel

smoothing parameter

$$p(x) = \frac{1}{N}\sum_{i=1}^{N} \begin{cases} 1 - \lambda\,; X_i = x \\ \dfrac{\lambda}{c-1}\,; otherwise \end{cases}$$

# of categories

Bandwidth Selection for Categorical Variables

- For the estimator we selected, we can derive a plug-in estimate of the bandwidth

$$\lambda = \frac{c-1}{c}\left(1 + \frac{n\sum_{x=1}^{c}\left[\frac{1}{c} - p(x)\right]^2}{\sum_{x=1}^{c} p(x)[1-p(x)]}\right)^{-1}$$

Finding the Optimal Bandwidth

```
def find_bandwidth(data):
    c = len(np.unique(data))
    n = len(data)
    _ , counts = np.unique(data, return_counts=True)
    probabilities = counts/sum(counts)

    numerator = n*sum([(1/c + p)**2 for p in probabilities])
    denominator = sum([p*(1-p) for p in probabilities])
    return (c-1)/c * (1+numerator/denominator)**-1

print(find_bandwidth(sales))
>>> 0.3530903534154827
```

Solving the Example

Histogram should match red dot

Sm. VSF
-> constant or not