# Density Estimation

EN5422/EV4238 | Fall 2023

w06_density_1.pdf

(Week 6 – 1/2)

## Contents

# 1 Density Estimation Intro

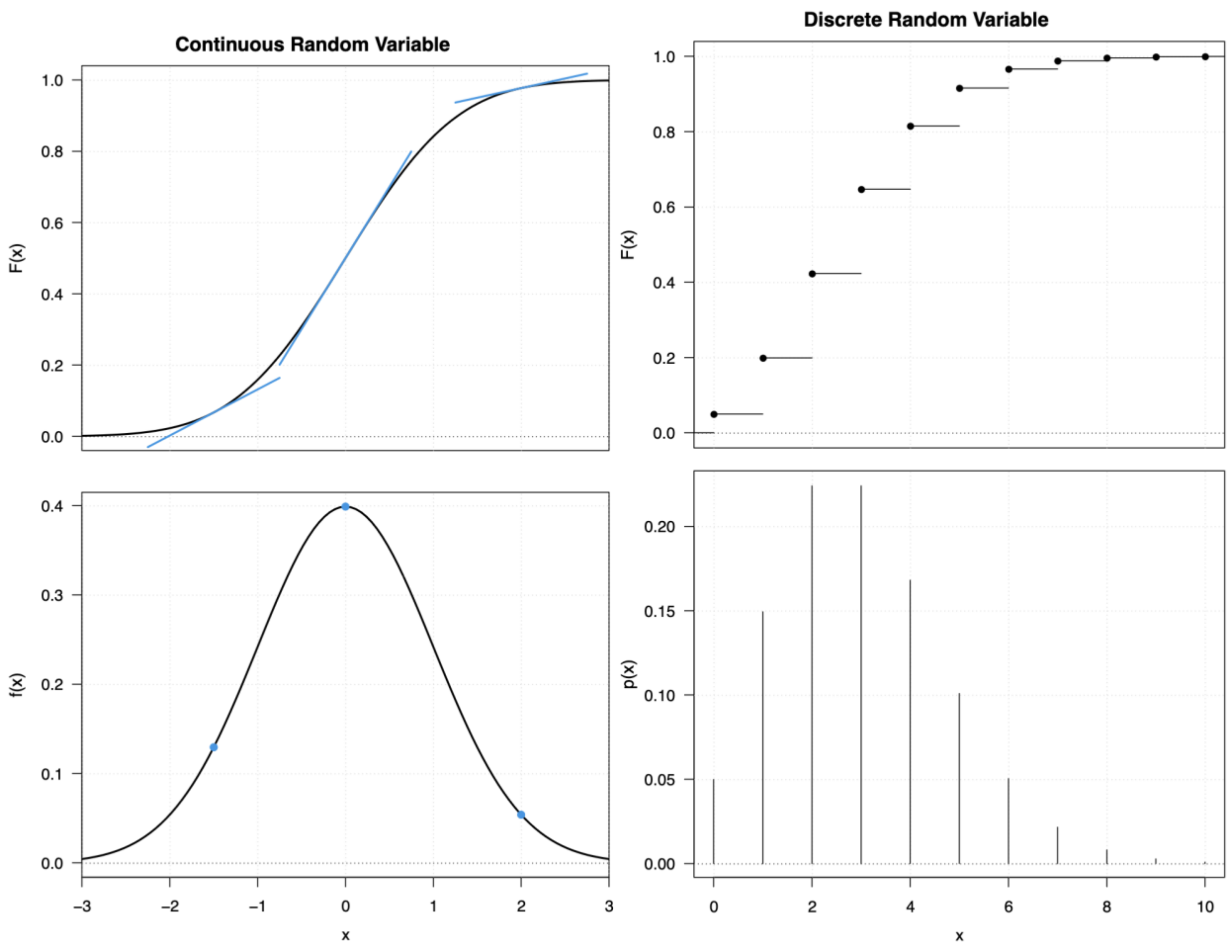
## 1.1 Distributions

- For many problems, so optimal decision can be formulated if we know the distribution of the relevant random variable(s).
    - The random variable(s) are the unknown or unobserved values.
- Often, only certain properties of the distribution (expected value, variance, quantiles) are needed to make decisions.
- Much of statistics is involved with estimation of the distributions or their properties.

### 1.1.1 Random Variables

Let $X$ be a **random variable** of interest.

- The **cumulative distribution function (cdf)** is $F(x) = \Pr(X \leq x)$.
    - $F(x)$ is the probability that the random variable $X$ ("big X") will take a value less than or equal to $x$ ("little x").
- For *discrete* random variables, the **probability mass function (pmf)** is $f(k) = \Pr(X = k)$.
    - $f(k) \geq 0, \sum_k f(k) = 1$
    - $f(k) = F(k) - F(k-1)$
- For *continuous* random variables, the **probability density function (pdf)** is $f(x) = \frac{d}{dx}F(x)$.
    - $f(x) \geq 0, \int_{-\infty}^{\infty} f(x) = 1$



### 1.1.2 Parametric Distributions

A **parametric** distribution, $f(x;\boldsymbol{\theta})$ is one that is fully characterized by a set of parameters, $\boldsymbol{\theta}$. Examples include:

- Normal/Gaussian
  - parameters: mean $\mu$, standard deviation $\sigma$
- Poisson
  - parameter: rate $\lambda$
- Binomial
  - parameters: size $n$, probability $p$
- There are also multivariate version: Gaussian $N(\mu, \Sigma)$

If we can model (assume) the random variable follows a specific parametric distribution, then we only need to estimate the parameter(s) to have the entire distribution characterized. The parameters are often of direct interest themselves (mean, standard deviation).

| **Note** |
|---|
| More details about some common parametric distributions can be found in the PyMC document. |

### 1.1.3 Non-Parametric Distributions

A distribution can also be estimated using **non-parametric** methods (e.g., histograms, kernel methods, splines). These approaches do not enforce a parametric family (which is essentially a type of prior knowledge), but let the data *fully* determine the shape of the density/pmf/cdf. As you might imagine more data is required for these methods to work well. Non-parametric approaches are excellent for exploratory data analysis, but can also be very useful for other types of modeling (e.g., classification, anomaly detection). **All in all, non-parametric methods aim to estimate the distribution directly from the data without making strong assumptions.** non-parametric methods offer flexibility by avoiding strong distributional assumptions. While they can require more data and can be computationally intensive, their adaptability makes them valuable tools in a wide range of applications.

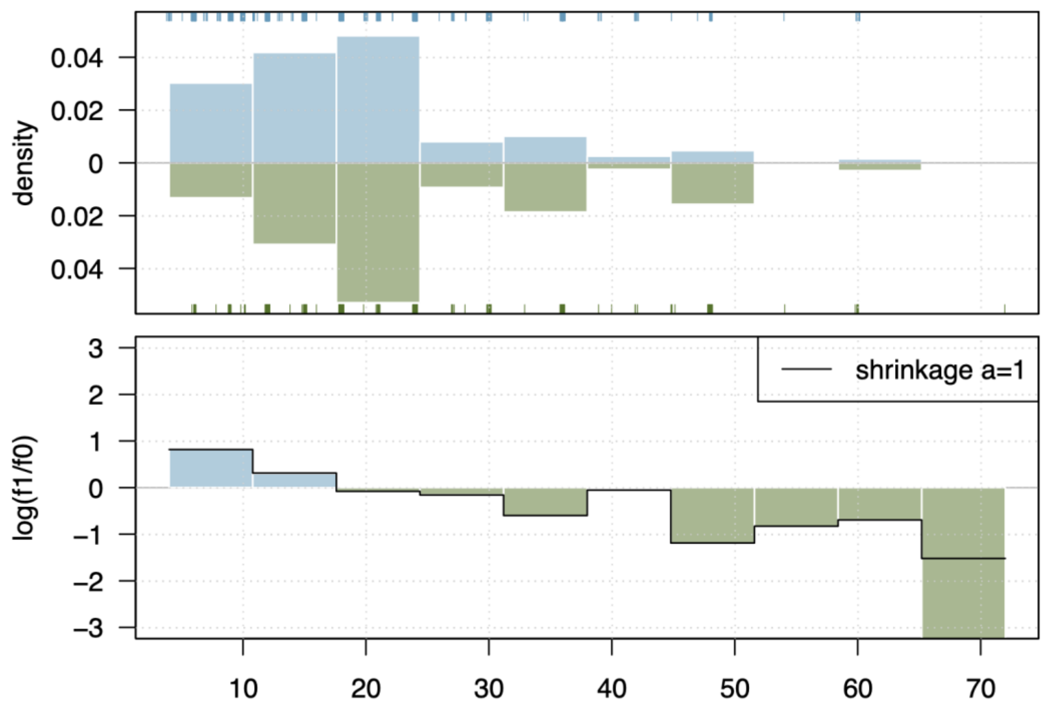
| **Note** |
|---|
| Because non-parametric methods don't make strong assumptions about the underlying distribution, they often require more data to provide accurate estimates. With too little data, a non-parametric estimate might be overly influenced by noise or random variations. |

## 1.2 Example: Default Classification

Density estimation can be useful in *classification problems*, where the goal is to determine which class a new observation belongs to.

Below are two *histogram* density estimates; one for customers of a German bank that have good credit (blue) and the other for customers who defaulted (green). If a new customer is observed to have $X = 5$, then the evidence favors them having good credit because $X = 5$ is more likely under customers with good credit.

The bottom plot shows the corresponding log density ratio, which can help the bank make a decision on the customer’s credit-worthiness.



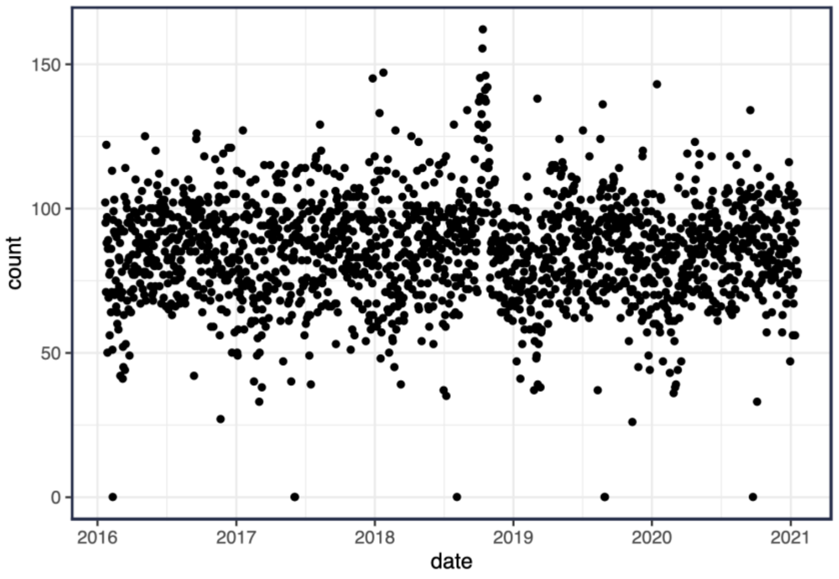
## 1.3 Example: Association Analysis

Density estimation can be useful in *association analysis*, where the goal is to find the regions with unusually high density (bump-hunting).

## 1.4 Example: Disease Outbreak Detection

Density estimation can be useful in *anomaly detection systems*, where the goal is to (often quickly) determine the time point when observations starting coming from a new or different distribution.

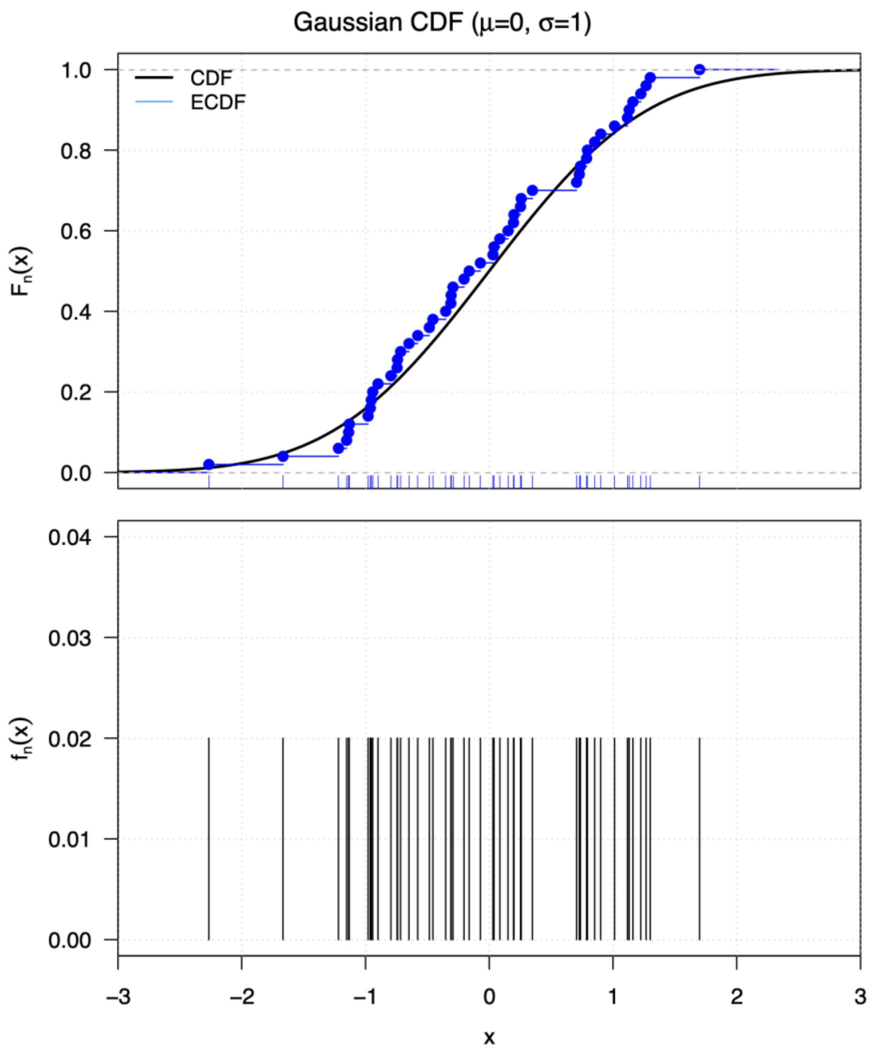
Below is simulated disease outbreak data representing the number of cases of some reported symptoms in an Emergency Department. If we can estimate the distribution of the baseline, or normal counts, on each day, then we will be able to flag an anomaly whenever the observations become *unlikely*.



## 1.5 Estimation

- These problems would be relatively easy to solve if we knew the exact distributions of the random variables of interest.
- Unfortunately, this is usually never the case (but of course: flipping coins, drawing cards, and playing with urns is different).
- We must use data to estimate the aspects/parameters of a distribution necessary to make good decisions.
  - And it is important to be mindful of the resulting uncertainty (bias, variance) in our estimation.

### 1.5.1 Empirical CDF and PDF



Data: $n = 50$ from $X \sim N(0, 1)$

## 2 Parametric Density Estimation

### 2.1 Method of Moments Estimation (MOM)

- Let $X$ be a random variable with *pdf/pmf* $f(x;\theta)$ parameterized by $\theta \in \Theta$.
- Let $D = \{X_1, X_2, \ldots,\ X_n\}$ be the observed data.
- *Method of Moments (MOM)* estimators match the sample moments to the theoretical moments.
  - This works when the parameter(s) can be written as functions of the moments.
  - To estimate *p* parameters, use *p* moments.
- 1st moments
  - The 1st *theoretical* moment $E[X] = \sum_i x_i P(X = x_i)$ is the mean.
  - The 1st *sample* moment is the sample mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} X_i$
  - If $g_1(\theta) = E[X]$, then set $g_1(\theta_{MM}) = \bar{x}$ and solve for $\theta_{MM}$.
- 2nd (central) moments
  - The 2nd *theoretical* central moment $V(X) = E[(X-\mu)^2]$ is the variance.
  - The 2nd *sample* central moment is the sample variance $\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2$.

**Example**

The **Poisson distribution** ($f(x) = \frac{\lambda^x}{x!}e^{-\lambda}$) is defined by a single parameter, $\lambda$, which represents both its mean and its variance.

**Theoretical Moment**: For the Poisson distribution, the expected value (or the first moment) is: $E[X] = \lambda$

**Toy Data**: Let's say we have a small sample of data points: X={2,3,3,4,2}.

**Sample Moment**: The sample mean $\bar{x}$ is $\frac{1}{n}\sum_{i=1}^{n} X_i$=2.5.

**Method of Moments Estimation**: For the **Poisson distribution**, using the method of moments, we had set: $\lambda_{MM} = 2.8$

So, using the MOM with our toy data, we would estimate $\lambda$ (the parameter for the Poisson distribution) to be 2.8. Note that the MOM provides just one way to estimate parameters. Other methods like Maximum Likelihood Estimation (MLE) might give different estimates, and depending on the context, one method might be preferable over the other.

**Note**

The sample mean is a statistic, which is a function of the sample data. It provides an estimate of the expected value, but it might not be the exact value, especially for small sample sizes. As the sample size grows, the sample mean will generally converge to the true expected value of the distribution (this is known by the law of large numbers). In essence, the theoretical moment is a property of the distribution, and the sample moment is an estimate of this property based on observed data. They are treated separately because one is a characteristic of the theoretical distribution and the other is derived from actual data.

- Maximum Likelihood estimation usually produces a *better* estimate, so we will focus on the MLE approach.

## 2.2 Maximum Likelihood Estimation (MLE)

- Let $X$ be a random variable with *pdf/pmf* $f(x;\theta)$ parameterized by $\theta \in \Theta$.
- Let $D = \{X_1, X_2, \dots, X_n\}$ be the observed data.
- *Maximum Likelihood Estimation (MLE)* uses the value of $\theta$ that maximizes the *likelihood*:

$$L(\theta) = P(X_1, X_2, \dots, X_n; \theta)$$

- The likelihood is written as a function of $\theta$ and treating the observed data as known
- When the observations are *independent*, this becomes:

$$L(\theta) = \prod_{i=1}^{n} P(X_i;\theta) = \prod_{i=1}^{n} f(x_i;\theta) = \prod_{i=1}^{n} f(x_i;\theta)$$

- The log-likelihood (under independence) becomes:

$$\log L(\theta) = \sum_{i=1}^{n} \log f(x_i;\theta)$$

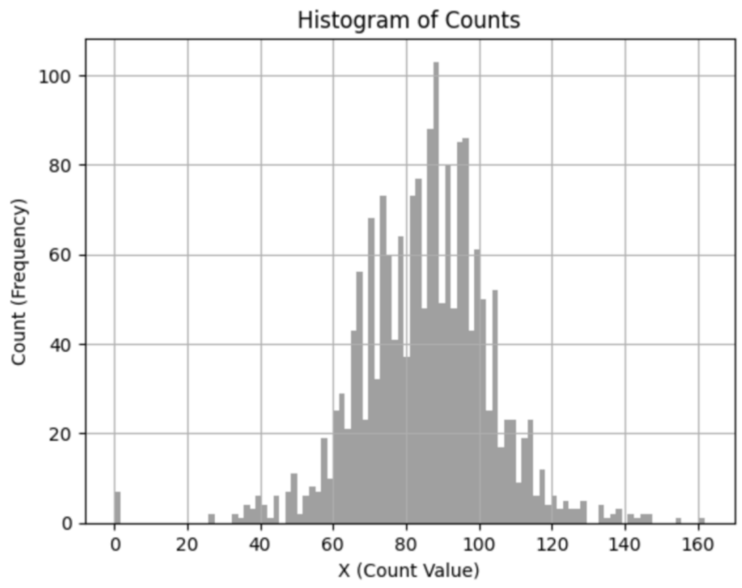
- And the MLE is:

$$\theta_{MLE} = \arg\max_{\theta\in\Theta} L(\theta) = \arg\max_{\theta\in\Theta} \log L(\theta)$$

**Your Turn #1**

In the disease outbreak example, we needed to estimate the distribution of reported symptoms of some disease *on a normal day*. Then *unusual or rare* counts could be considered anomalous and a potential indication of a disease outbreak or bio-attack.

Estimate the base line density of ER counts.



1. Use a Poisson model.
2. Use a Negative Binomial Model.
3. Use a Gaussian model.
4. What is the probability that we would get more than > 150 or < 50 counts on a *regular* day?

Note: Distribution Reference Sheet

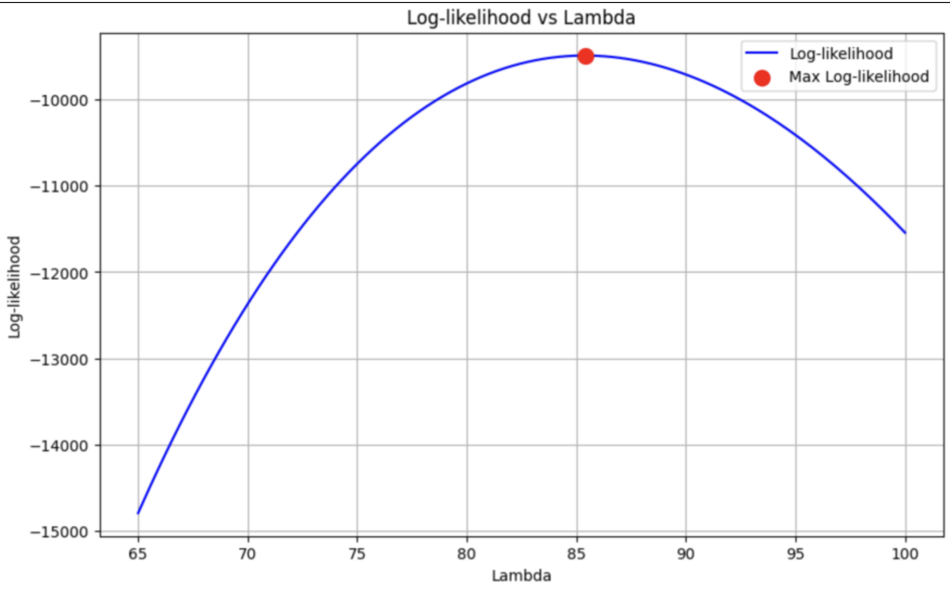
### 2.2.1 Poisson MLE: Grid Search

- Notation:
  - $X \sim Pois(\lambda)$
  - $\Pr(X = x) = f(x; \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$ Where $f(x)$ is *pmf* and $x = \{0,1,\ldots\}$.
  - $E[X] = V[X] = \lambda$

**Grid Search**

```
from scipy.stats import poisson

# Get sequence of lambda values
lam_seq = np.linspace(65, 100, 200)
nlam = len(lam_seq)

# Calculate the log-likelihood for those lambda values
loglike = np.array([np.sum(poisson.logpmf(x, mu=lam)) for lam in lam_seq])

# Find the lambda value that maximizes the log-likelihood
max_index = np.argmax(loglike)
best_lambda = lam_seq[max_index]
max_loglike = loglike[max_index]
```

```
# Plotting the results
plt.figure(figsize=(10, 6))
plt.plot(lam_seq, loglike, label='Log-likelihood', color='blue')
plt.scatter(best_lambda, max_loglike, color='red', s=100, zorder=5,
label='Max Log-likelihood')  # Red dot marking the maximum
plt.xlabel('Lambda')
plt.ylabel('Log-likelihood')
plt.title('Log-likelihood vs Lambda')
plt.legend()
plt.grid(True)
plt.show()

> 85.402
```



- The grid search gives $\hat{\lambda} = 85.402$
  - Searched 200 values between 65 and 100.

### 2.2.2 Poisson MLE: Calculus

**Your Turn #2**

Derive the MLE using Poisson model using calculus.

**Poisson Distribution:** The probability mass function (PMF) of a Poisson distribution is given by:

$$P(X = k) = \frac{e^{(-\lambda \lambda^k)}}{k!}$$

where $\lambda$ is the rate parameter and $k$ is a non-negative integer.

**Likelihood Function:** Given a sample $x_1, x_2, \cdots, x_n$ from a Poisson distribution, the likelihood function $L(\lambda)$ is:

$$L(\lambda) = \prod_{i=1}^{n} \frac{e^{-\lambda}\lambda^{x_i}}{x_i!}$$

Taking the natural logarithm, we get the log-likelihood function $l(\lambda)$:

$$l(\lambda) = \sum_{i=1}^{n} [-\lambda + x_i \ln(\lambda) - \ln(x_i!)]$$

**Differentiating the Log-Likelihood:** To find the MLE, we need to differentiate the log-likelihood with respect to λ and set the result equal to zero.

Differentiating:

$$\frac{dl(\lambda)}{d\lambda} = -n + \sum_{i=1}^{n} \frac{x_i}{\lambda}$$

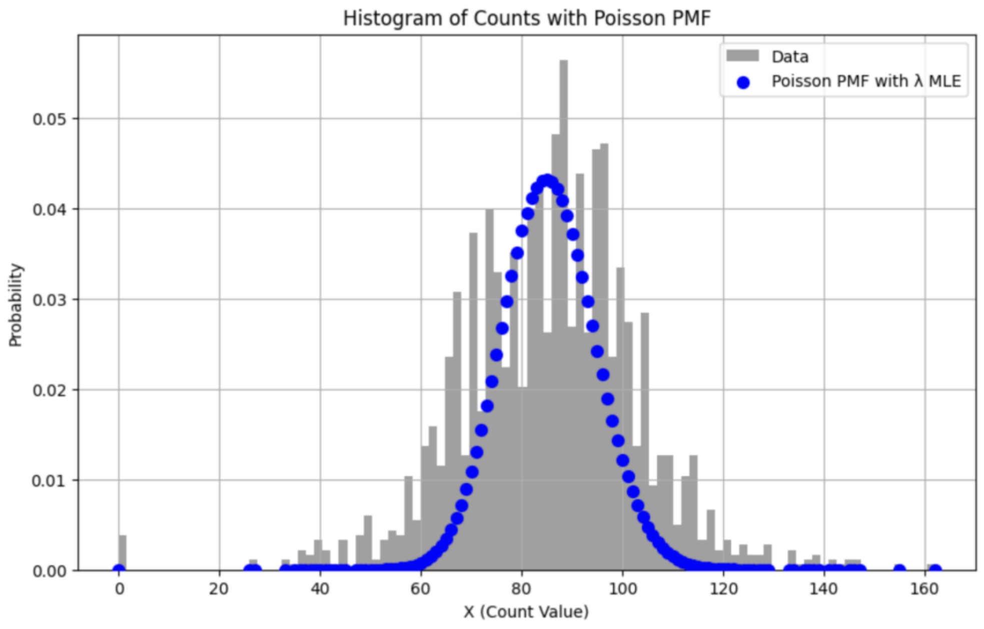
Setting this equal to zero:

$$-n + \sum_{i=1}^{n} \frac{x_i}{\lambda} = 0$$

From the above equation, rearranging gives the MLE $\hat{\lambda}$ of $\lambda$:

$$\frac{1}{n}\sum_{i=1}^{n} x_i = \hat{\lambda}$$

## Estimated *pmf* using Poisson MLE



**Note**

- Does the Poisson model look like a good one?
- Where do you see lack of fit?

### 2.2.3 Poisson MLE: using statmodels Python library

```
import statsmodels.api as sm
# Fit the Poisson model
poisson_model = sm.GLM(np.asarray(x), np.ones_like(x),
family=sm.families.Poisson()).fit()
# Get the coefficient (which is equivalent to the lambda in the Poisson
distribution)
lambda_value = np.exp(poisson_model.params[0])
print(f'lambda: {lambda_value:.2f}')
```

### 2.2.4 Negative Binomial: MLE

- The Negative Binomial distribution can help for modeling count data with obverdispersion
- Notation:
  - $X \sim N\ Bin(\mu, r)$ (using *mean* parameterization)
  - $\mu > 0, r > 0$
  - $E[X] = \mu, V[X] = \mu + \frac{\mu^2}{r}$
  - Mean representation: link

$$\Pr(X = x; r, \mu) = \left(\frac{\Gamma(r + x)}{x!\ \Gamma(r)}\right) * \left(\frac{r}{r + \mu}\right)^r * \left(\frac{\mu}{r + \mu}\right)^x$$

  - $n! = \Gamma(n - 1)$
  - $\Gamma(n) = \int_0^\infty e^{-x} x^{n-1} dx$

**Your Turn #3**

Use MLE to estimate $\theta = (r, \mu)$.

**Negative Binomial Distribution:** The probability mass function (PMF) of a N. Binomial Dist. is given by:

$$\Pr(X = x; r, \mu) = \left(\frac{\Gamma(r + x)}{x!\ \Gamma(r)}\right) * \left(\frac{r}{r + \mu}\right)^r * \left(\frac{\mu}{r + \mu}\right)^x$$

**Likelihood Function:** Given a sample $x_1, x_2, \cdots, x_n$from a N. Binomial Dist., the likelihood function $L(r, \mu)$ is the product of the individual probabilities:

$$L(r,\mu) = \prod_{i=1}^{n} \Pr(X = x;\ r,\mu)$$

Taking the natural logarithm, we get the log-likelihood function $l(r,\mu)$:

$$l(r,\mu) = \ln\big(L(r,\mu)\big)$$

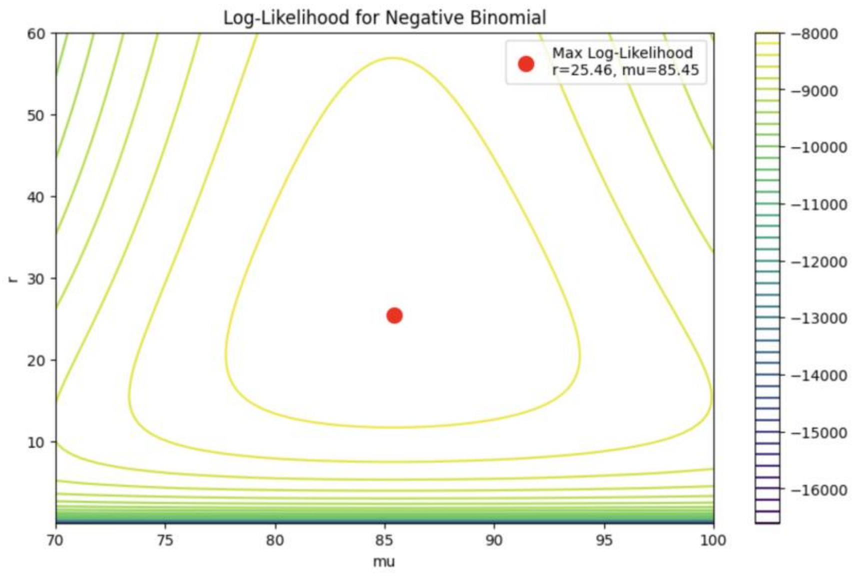
**Differentiating the Log-Likelihood:** To find the MLE, we need to differentiate the log-likelihood with respect to $r,\mu$ and set the result equal to zero. This will give you tow equations, one for $r$ and one for $\mu$.

Solving for $r$ and $\mu$.

$$\frac{\partial l(r,\mu)}{\partial r} = 0$$

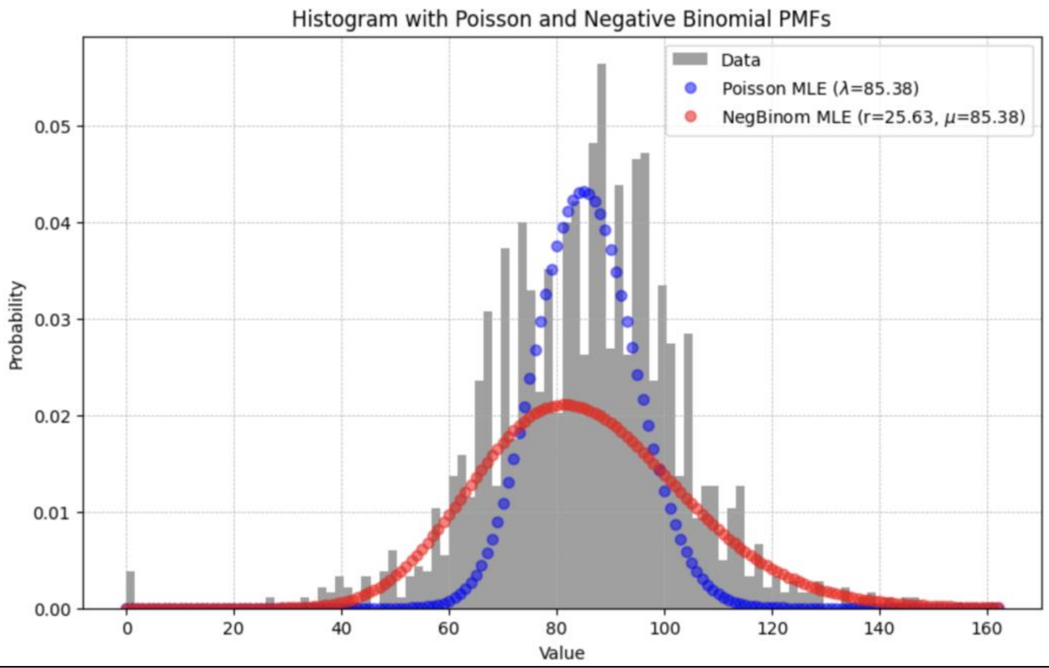
$$\frac{\partial l(r,\mu)}{\partial \mu} = 0$$

The solutions to these equations are the maximum likelihood estimates of $r$ and $\mu$. However, these equations are usually quite complicated and may not have a simple closed-form solution. In practice, numerical methods, such as the Newton-Raphson method, might be used to solve them. For more information check this link.



- Use Python scipy library to find these parameters

```
from scipy.stats import nbinom
from scipy.optimize import minimize

# Function to compute negative log-likelihood for Negative Binomial
def nll(params, data):
    r, log_mu = params
    mu = np.exp(log_mu)  # We optimize in the log space for mu for stability
    prob = r / (r + mu)
    ll = nbinom.logpmf(data, r, prob)
    return -ll.sum()

# Initial guess
init_params = [10, np.log(np.mean(x))]

# Minimize the negative log-likelihood
result = minimize(nll, init_params, args=(x), bounds=((0.01, 1000),
(np.log(1), np.log(200))))

r_mle, log_mu_mle = result.x
mu_mle = np.exp(log_mu_mle)

print(f"MLE for size (r): {r_mle:.2f}")
print(f"MLE for mu: {mu_mle:.2f}")
```



| Note |
|---|
| • Does the Negative Binomial model look better than Poisson?<br>• Are there any remaining concerns? |

### 2.2.5 Example: Gaussian/Normal

- Data are non-negative integers, not continuous, so Gaussian is clearly “wrong”. But as the famous saying goes:

“All models are wrong, but some are useful”. – George E. P. Box

- Notation:
  1. $X \sim N(\mu, \sigma)$
  2. $\mu \in \mathbf{R}, \sigma > 0$
  3. $E[X] = \mu, V[X] = \sigma^2$

**Your Turn #4**

Find the MLE for $(\mu, \sigma)$

Given a sample $x_1, x_2, \cdots, x_n$ from a normal distribution with unknown mean $\mu$ and unknown variance $\sigma^2$, the likelihood function for the sample is given by:

$$L(\mu, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

The log-likelihood function is the natural logarithm of the likelihood:

$$l(\mu, \sigma^2) = \sum_{i=1}^{n} \left( -\frac{1}{2} \ln(2\pi\sigma^2) - \frac{(x_i - \mu)^2}{2\sigma^2} \right)$$

To find the MLE for $\mu$, differentiate the log-likelihood with respect to $\mu$ and set it to zero:

$$\frac{\partial l}{\partial \mu} = \sum_{i=1}^{n} \frac{x_i - \mu}{\sigma^2} = 0$$

From the above equation, we get:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Which is just the sample mean.

Next, find the MLE for $\sigma^2$, differentiate the log-likelihood with respect to $\sigma^2$ and set it to zero:

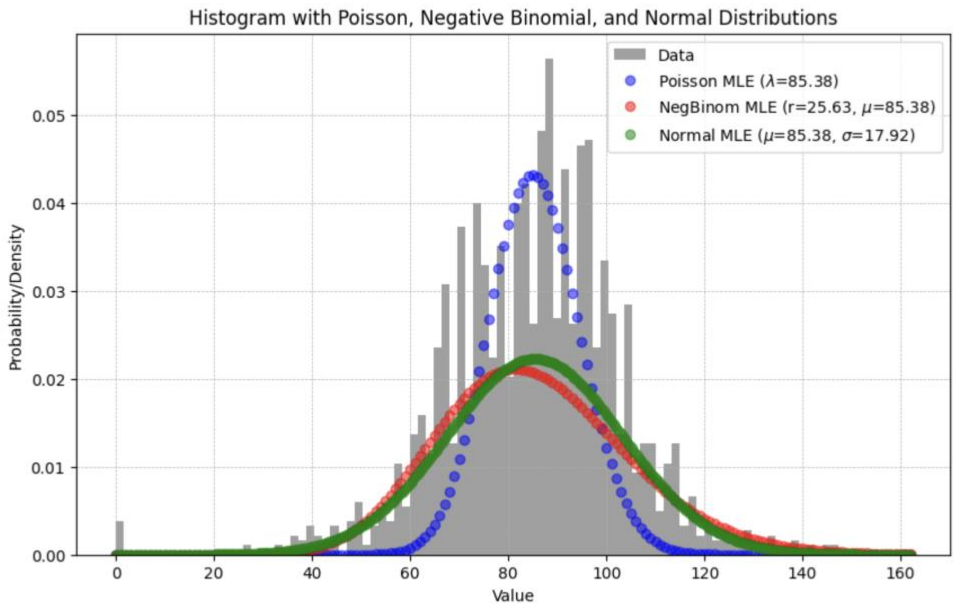
$$\frac{\partial l}{\partial \sigma^2} = \sum_{i=1}^{n} \left( -\frac{1}{2\sigma^2} + \frac{(x_i - \mu)^2}{2\sigma^4} \right) = 0$$

From the above equation, we get:

$$\widehat{\sigma^2} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})^2$$

Which is just the sample variance. Therefore, the MLE for $\mu$ is the sample mean and the MLE for $\sigma^2$ is the sample variance.

### 2.2.6 Comparison of Models



- Models:
  1. Poisson: $\lambda = 85.382$
  2. Neg.Binon: $\mu = 85.386, r = 25.627$
  3. Gaussian: $\mu = 85.382, \sigma = 17.919$

**Your Turn #5**

Which model do you choose? Why?

### 2.2.7 Density Evaluation

1. AIC/BIC
2. Cross-validation
   • need a loss function (e.g., negative log-likelihood)
   • can consider other Goodness-of-Fit (GOF) metrics

## 2.3 Bayesian Estimation

In Bayesian analysis, the parameters(s) themselves are treated as *random variables*.

- In MLE, the parameters are assumed fixed, but unknown.

Prior knowledge, which is any information known about the parameter(s) *before the data are seen*, is captured in the *prior distribution.*

- Let $g(\theta)$ be the (possibly multivariate) prior pmf/pdf

Bayes theory gives us the *posterior distribution*,

$$f(\theta \mid D) = \frac{P(D \mid \theta)g(\theta)}{\int_{\theta \in \Theta} P(D \mid \theta)g(\theta)\mathrm{d}\theta}$$

- $P(D \mid \theta) = P(X_1, X_2, \dots, X_n \mid \theta) =$ likelihood
- $\int_{\theta \in \Theta} P(D \mid \theta)g(\theta)\mathrm{d}\theta = P(D)$ is the *normalizing constant* (not function of $\theta$)
- $f(\theta \mid D)$ is the *posterior distribution*, which contains the updated knowledge about the parameter(s).

### 2.3.1 Bayesian Point Estimation

1. Posterior Mean
$$\hat{\theta}_{\mathrm{PM}} = E[\theta \mid D] = \int_{\theta \in \Theta} \theta f(\theta \mid D)\mathrm{d}\theta$$

2. MAP (Maximum a posteriori)

$$\begin{aligned}\hat{\theta}_{\mathrm{MAP}} &= \arg\max_{\theta \in \Theta} f(\theta \mid D) \\ &= \arg\max_{\theta \in \Theta} P(D \mid \theta)g(\theta) \\ &= \arg\max_{\theta \in \Theta} (\log P(D \mid \theta) + \log g(\theta))\end{aligned}$$

# 3 Non-Parametric Density Estimation

## 3.1 Non-Parametric Density Estimation

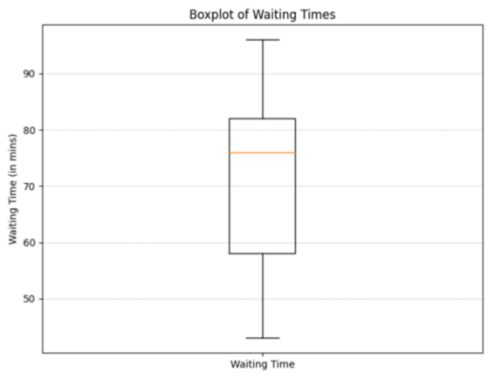
The old faithful geyser in Yellowstone National Park is one of the most regular geysers in the park. The waiting time between eruptions is between 35 and 120 mins.

Live Streaming Webcam with eruption predictions

Because the nearby Yellowstone Lodge is nice and warm in the winter, and serves good ice cream in the summer, you may be distracted from stepping outside to watch the eruption. Let's see if we can determine the best time to leave the cozy lodge and go outside to watch the next eruption.

```python
import statsmodels.api as sm
import numpy as np

# Load the Old Faithful data
faithful_dataset = sm.datasets.get_rdataset("faithful", "datasets")
faithful_data = faithful_dataset.data
wait = faithful_data['waiting'].values

# Calculate summary stats
sample_size = len(wait)
print(f"Sample size: {sample_size}")

# Six number summary
min_wait = np.min(wait)
max_wait = np.max(wait)
q1 = np.percentile(wait, 25)
q3 = np.percentile(wait, 75)
median_wait = np.median(wait)
mean_wait = np.mean(wait)

print(f"Min.: {min_wait}\n1st Qu.: {q1}\nMedian: {median_wait}\nMean:
{mean_wait}\n3rd Qu.: {q3}\nMax.: {max_wait}")

# Mean
print(f"Mean: {mean_wait:.2f}")

# Standard deviation
std_dev = np.std(wait)
print(f"Standard Deviation: {std_dev:.2f}")

# Median
print(f"Median: {median_wait:.2f}")

# Quantiles
print(f"Quantiles:\n25%: {q1}\n50%: {median_wait}\n75%: {q3}")
```

```
Sample size: 272
Min.: 43
1st Qu.: 58.0
Median: 76.0
Mean: 70.8970588235294
3rd Qu.: 82.0
Max.: 96
Mean: 70.90
Standard Deviation: 13.57
Median: 76.00
Quantiles:
25%: 58.0
50%: 76.0
75%: 82.0
```



## Your Turn #6: Old Faithful

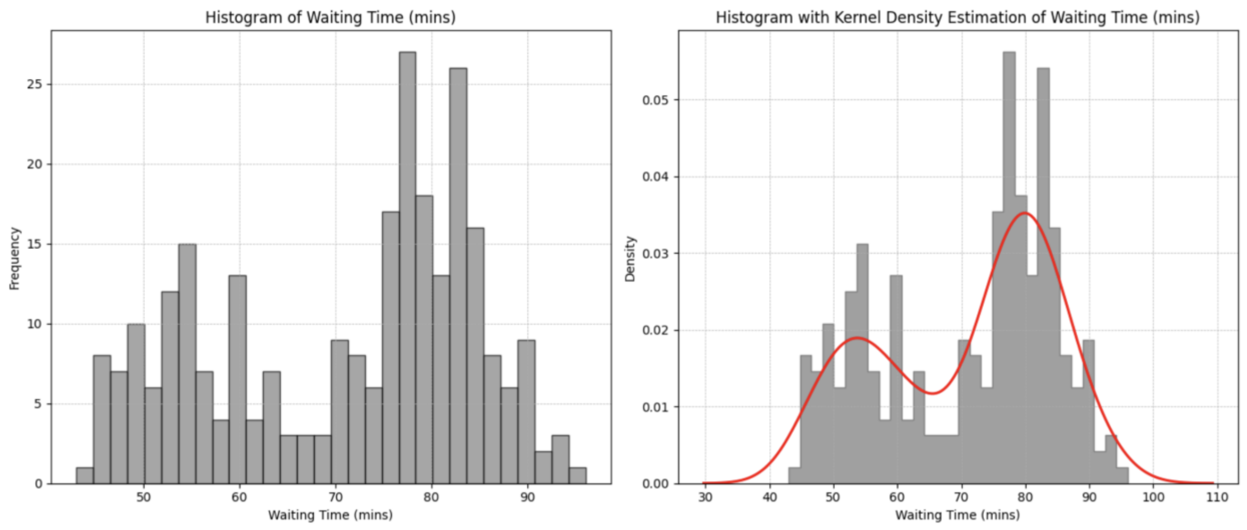
The data, summary statistics, and plots below represent a sample of *waiting times*, the time (in min) between Old Faithful eruptions.

1. What can you say about the shape of the distribution?
2. Would a Gaussian (i.e., Normal) Distribution be a good choice for modeling the distribution of these data?
3. What would you recommend?

More detail can be obtained by examining the histograms:

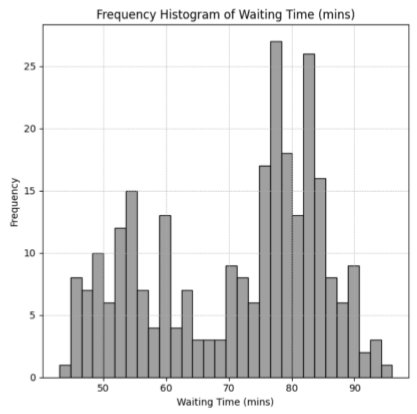
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Setup the 1x2 subplot grid
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 6))

# Histogram for 'waiting time' on subplot (1,1)
axes[0].hist(wait, bins=30, alpha=0.7, color='grey', edgecolor='black')
axes[0].set_title("Histogram of Waiting Time (mins)")
axes[0].set_xlabel("Waiting Time (mins)")
axes[0].set_ylabel("Frequency")
axes[0].grid(True, which="both", linestyle="--", linewidth=0.5)

# Overlaid histogram for 'waiting time' on subplot (1,2)
sns.histplot(wait, ax=axes[1], bins=30, color='grey', element="step",
stat="density", common_norm=False)
sns.kdeplot(wait, ax=axes[1], color="red", lw=2)
axes[1].set_title("Histogram with Kernel Density Estimation of Waiting Time
(mins)")
axes[1].set_xlabel("Waiting Time (mins)")
axes[1].set_ylabel("Density")
axes[1].grid(True, which="both", linestyle="--", linewidth=0.5)
```

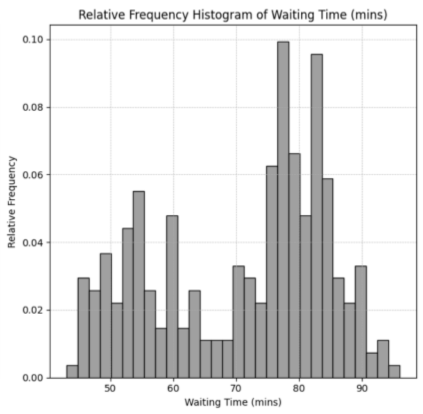
```
# Adjust the layout
plt.tight_layout()
plt.show()
```

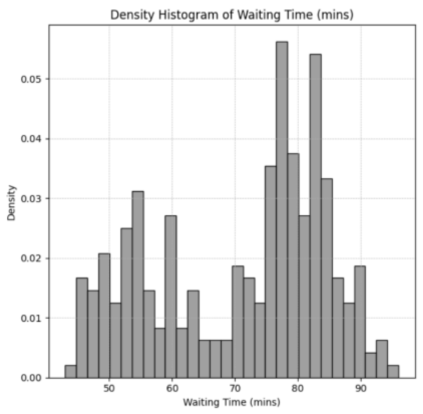


## 3.2 Histograms

- A histogram is a visual representation of *piece-wise* constant function.
- There are three primary types of histograms: **frequency**, **relative frequency**, and **density**







## 3.3 Density Histograms

- A *density histogram* is a special type of histogram that has the property of being a proper pdf: non-negative and integrate to 1.
    - $f(x) \geq 0 \ \forall x$ and $\int f(x)dx = 1$
- Histograms estimate the density as a piecewise constant function.

$$\hat{f}(x) = \sum_{j=1}^{J} b_j(x)\hat{\theta}_j$$

Where $b_j(x) = 1(x \in bin_j)/h_j$ and

- $b_j = [t_j, t_{j+1})$
- $t_1 < t_2 < \cdots < t_J$ are the break points for the bins
- $h_j = [t_j, t_{j+1}) = t_{j+1} - t_j$ is the **bin width** of bin *j*
    - Bin widths do *not* have to be equal
- $bin_j \cap bin_k = \emptyset$

### 3.3.1 Estimating Density Histograms

- Observe data $D = \{X_1, X_2, \ldots, X_N\}$
- Specify bins
- Denote $n_j$ as the number of observations in bin *j*
- $\hat{\theta}_j = \hat{p}_j = n_j/N$ is the usual (MLE) estimate
    - Given binning, *multinomial distribution*

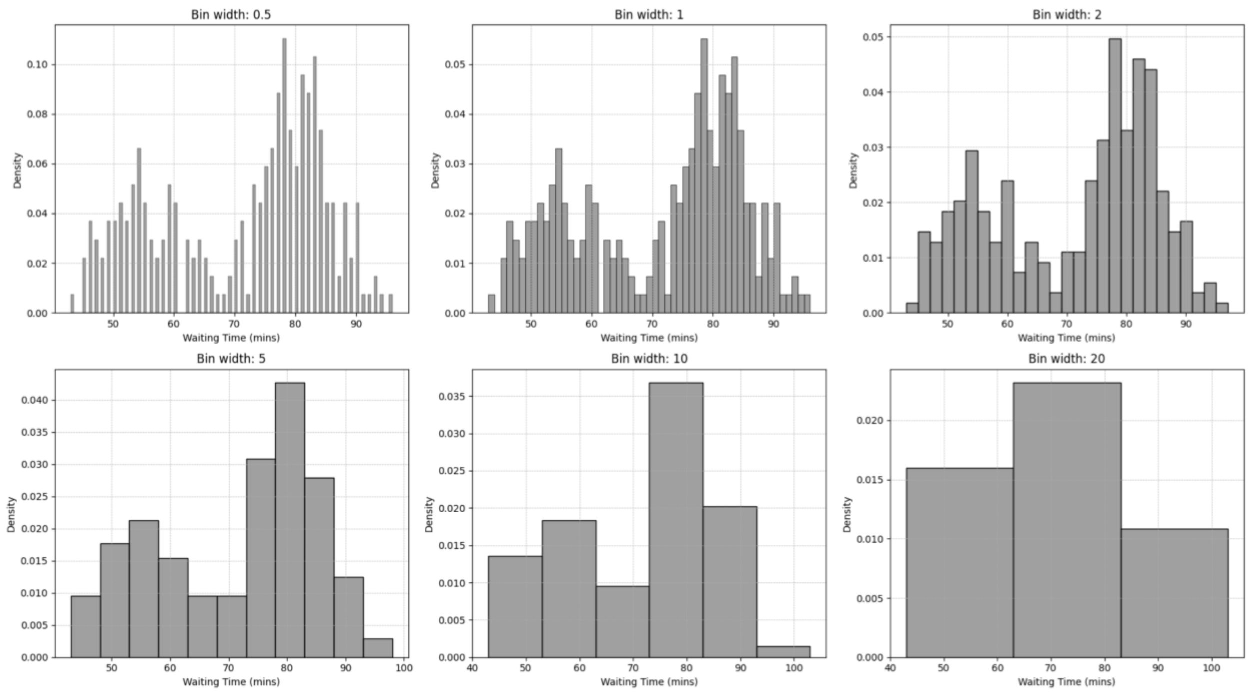
## 3.4 Local Properties of Histograms

- Histograms can be thought of as a *local* method of density estimation.

- Points are local to each other if they fall in the same bin
- Local is determined by bin break points.

- But this has some issues:
  - Some observations are “closer” to observations in a neighboring bin
  - Estimate is not *smooth* (but many true density functions ca often be assumed smooth)
  - **Bin shifts can have a big influence on the resulting density**

### 3.4.1 Old Faithful: Sensitivity to bin width



### 3.4.2 Histogram Neighborhood

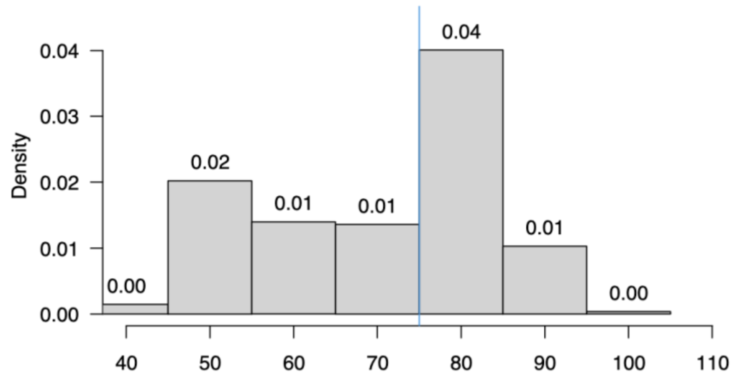
Consider estimating the density at a location *x*

- For a regular histogram (with bin width *h*), the MLE density is

$$\hat{f}(x) = \frac{n_j}{nh} \text{ for } x \in bin_j$$

which is a function of the number of observations in bin *j*.

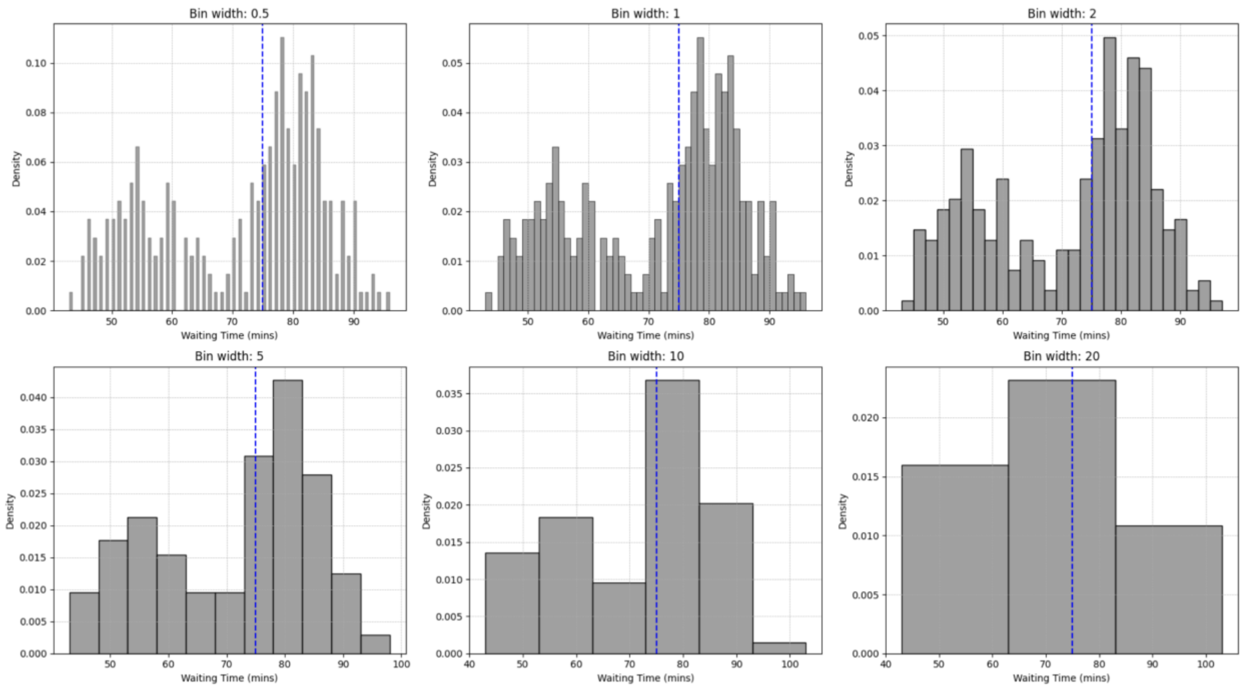
- But how do you feel if $x$ is close to the boundary (e.g., $x = 75$)?



### 3.4.3 Old Faithful: Sensitivity to bin shifts / anchor points

**Note**

What density should we use for at $x = 75$?



# 4 Kernel Density Estimation (KDE)

Kernel Density Estimation is an improvement on density histograms:

- Removes the bin anchor point parameter
- Allows more flexible definition of "neighborhood"

## 4.1 Local Density Estimation – Moving Window

Consider again estimating the density at a location *x*

- Regular Histogram (with midpoints $m_j$ and bin width *h*)

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1\left(|x_i - m_j| \leq \frac{h}{2}\right)}{h} \text{ for } x \in \text{bin } j$$

- Consider a moving window approach

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1\left(|x_i - x| \leq \frac{h}{2}\right)}{h}$$

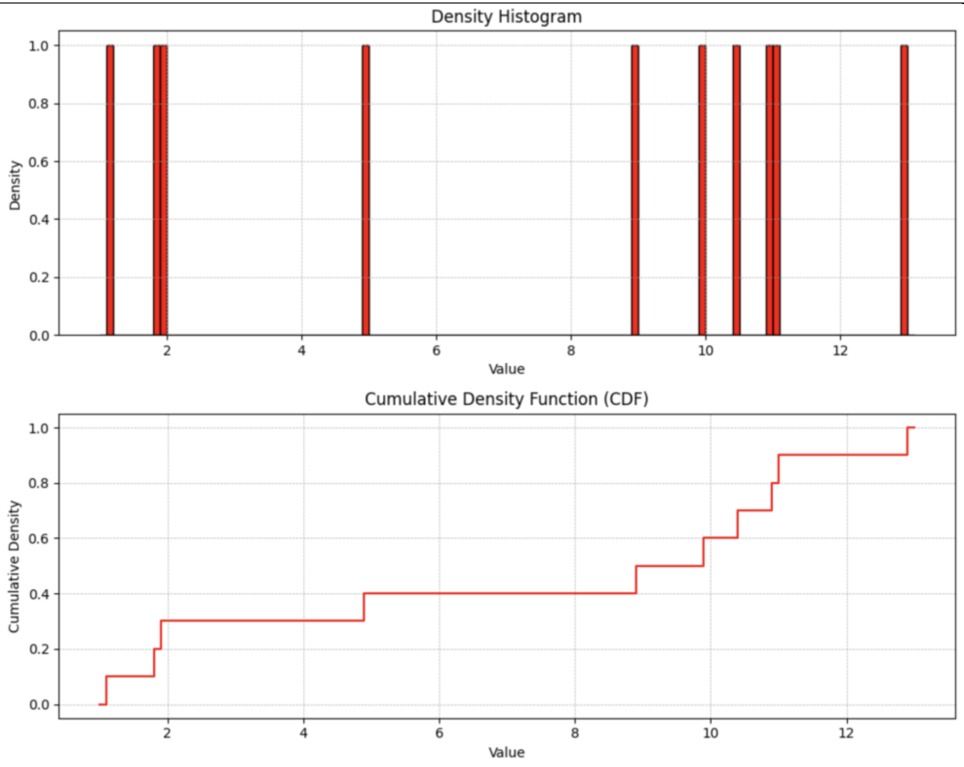
  This gives a more pleasing definition of local by centering a bin at *x*, to calculate the density at *x*.

- Equivalently, this estimates the derivative of empirical CDF (ECDF)

$$\hat{f}(x) = \frac{F_n\left(x + \frac{h}{2}\right) - F_n\left(x - \frac{h}{2}\right)}{h}$$

**Your Turn #7**

Consider a dataset $D = \{1.1, 1.9, 2, 5, 9, 10, 10.5, 11, 11.1, 13\}$. Use a *moving window* (*uniform kernel)* estimator to estimate the density at locations $x_0 = \{2, 7, 11\}$ using *h*=1.



The moving window (uniform kernel) estimator can be described as:

$$\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} I\left(\frac{|x - X_i|}{h} \le \frac{1}{2}\right)$$

Where:

- $\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} I\left(\frac{|x - X_i|}{h} \le \frac{1}{2}\right)$
- $n$ is the total number of data points.
- $h$ is the bandwidth.
- $I$ is an indicator function that takes the value 1 if the condition inside is true, and 0 otherwise.

Here are some popular kernel functions:

1. **Uniform (or Rectangular) Kernel**: This is the one we used. It is essentially a flat box.

$$K(u) = \begin{cases} \frac{1}{2} & \text{if } |u| \le 1 \\ 0 & \text{otherwise} \end{cases}$$

2. **Triangular Kernel**:

$$K(u) = \begin{cases} 1 - |u| & \text{if } |u| \le 1 \\ 0 & \text{otherwise} \end{cases}$$

3. **Epanechnikov Kernel**:

$$K(u) = \begin{cases} \frac{3}{4}(1 - u^2) & \text{if } |u| \le 1 \\ 0 & \text{otherwise} \end{cases}$$

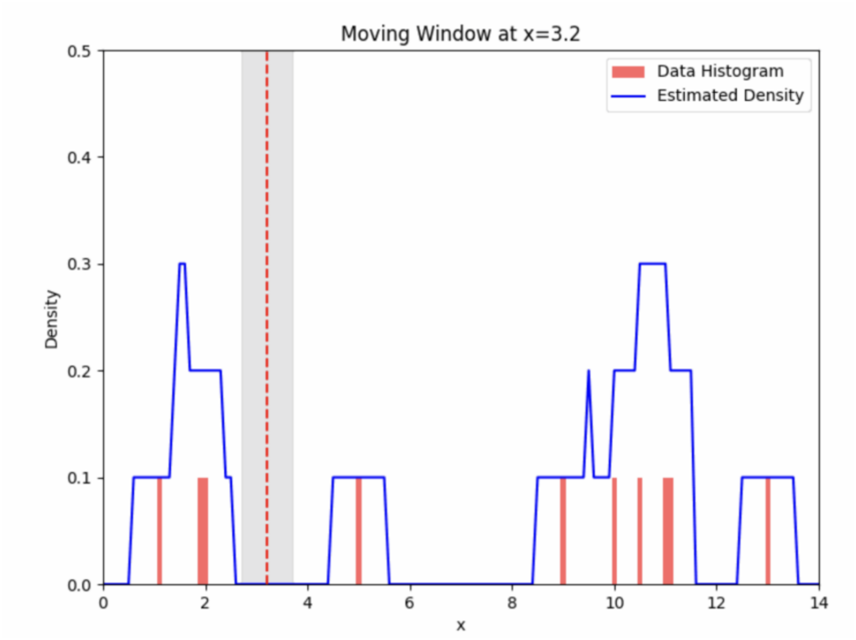
4. **Quartic (or Biweight) Kernel**:

$$K(u) = \begin{cases} \frac{15}{16}(1-u^2)^2 & \text{if } |u| \le 1 \\ 0 & \text{otherwise} \end{cases}$$

5. **Gaussian Kernel**: This is a very popular choice, and it results in a smoother density estimation.

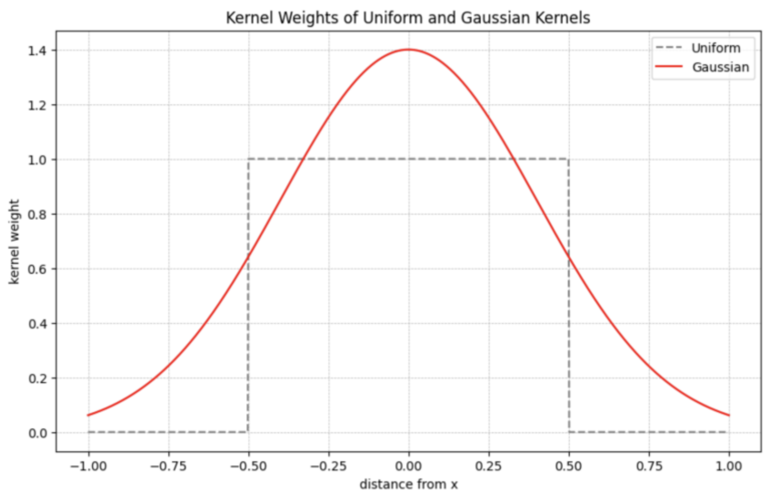
$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}$$

Demo of moving window (uniform) (gif)
Demo of moving window (Gaussian) (gif)



## 4.2 Kernels

- The moving window approach looks better than a histogram with the same bin width, but it is still not smooth.
- Instead of giving every observation in the window the same weight, we can assign a weight according to its distance from *x*



**Note**

1. **Uniform Kernel**: This is essentially a rectangle function that assigns a weight of 1 within the range and 0 outside of it.
2. **Gaussian Kernel**: This is the familiar bell-shaped curve that assigns weights based on the Gaussian function.

- More generally, the weights $K_h(u) = h^{-1}K\left(\frac{u}{h}\right)$ are called kernel functions
- Thus, a kernel density estimator is of the form

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x_i - x)$$

where the smoothing parameter $h$ is called the bandwidth and controls how fast the weights decay as a function of the distance from $x$

**Kernels**

There are several uses of *kernels* in this course. Generally speaking a kernel $K(x, y) \in \mathbf{R}$ is a measure of *similarity* between the two pints/vectors *x* and *y*.

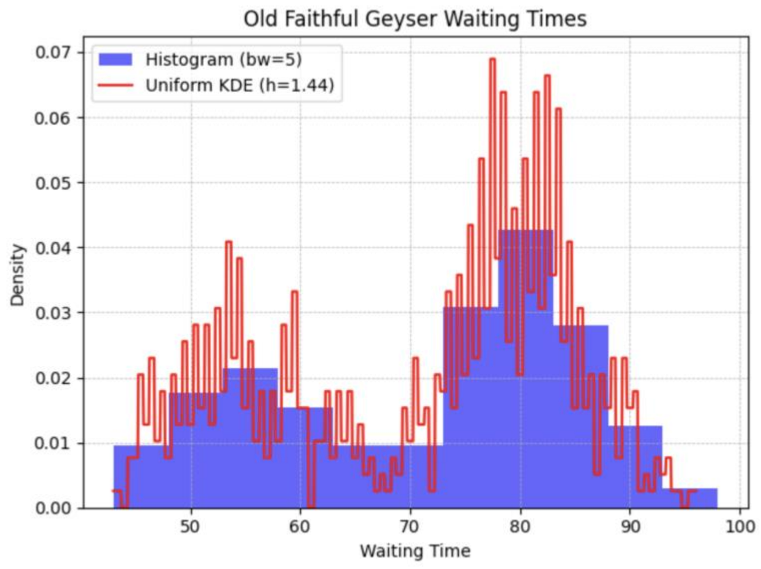
• For Kernel Density Estimation (KDE) its common to set the kernel to be a function of distance $K(|x-y|)$ or $K(||x-y||)$. $K(|x-y|)$ refers to the kernel function that uses the absolute difference between two data points *x* and *y*. This is used when dealing with one-dimensional data. The absolute difference, |*x*−*y*|, gives the "distance" between two points on a line. $K(||x-y||)$ refers to the kernel function that uses the Euclidean distance between two data points *x* and *y* in higher-dimensional space (like 2D, 3D, etc.). The notation ||x−y|| represents the norm (or length) of the vector difference between *x* and *y*, which is a general way to measure distance in multi-dimensional spaces.

• KDE is a non-parametric method to estimate the probability density function of a continuous random variable.

### 4.2.1 Uniform/Rectangular Kernel

- The moving window uses a *uniform* (or rectangular) kernel

$$K_h^{unif} = \frac{1\left(|x_i - x| \le \frac{h}{2}\right)}{h}$$

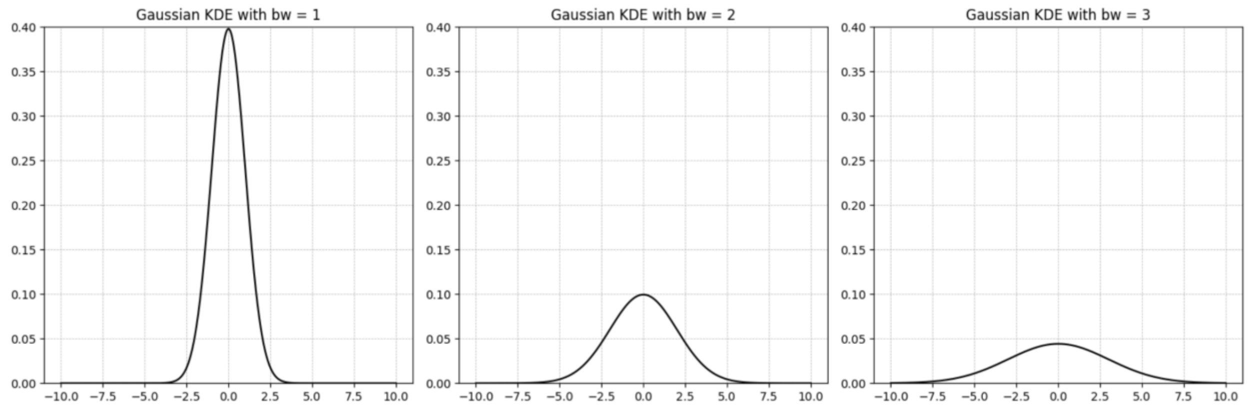


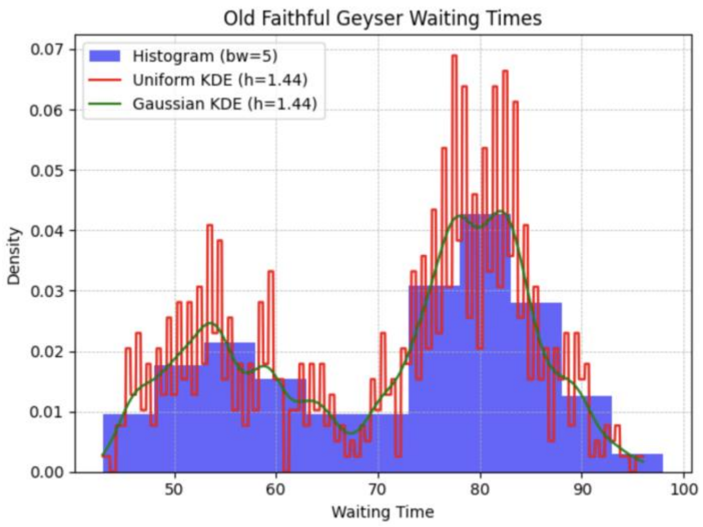
### 4.2.2 Gaussian/Normal kernel

- The most popular kernel is the **Gaussian kernel**

$$K_h^{gauss}(u) = \frac{1}{h\sqrt{2\pi}} \exp\left(-\frac{u^2}{2h^2}\right) = h^{-1}K\left(\frac{u}{h}\right) \text{ (where } K(\cdot) \text{ is standard normal pdf)}$$

Gaussian kernel
bw is standard deviation

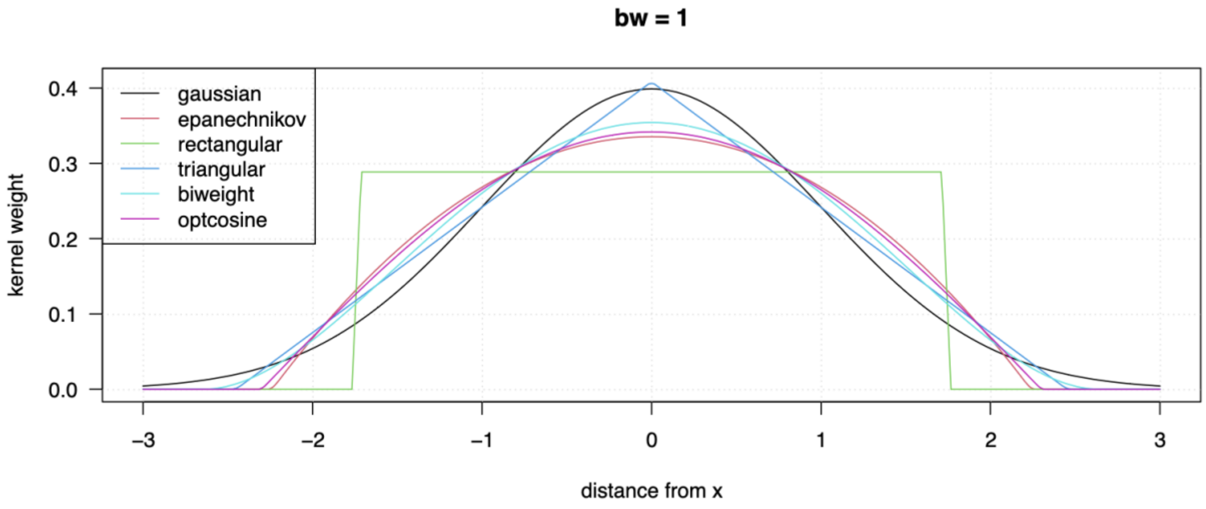




Note: [Demo of Gaussian KDE (gif)]

### 4.2.3 Other kernels

- There are many choices for kernels



### 4.2.4 Kernel Properties

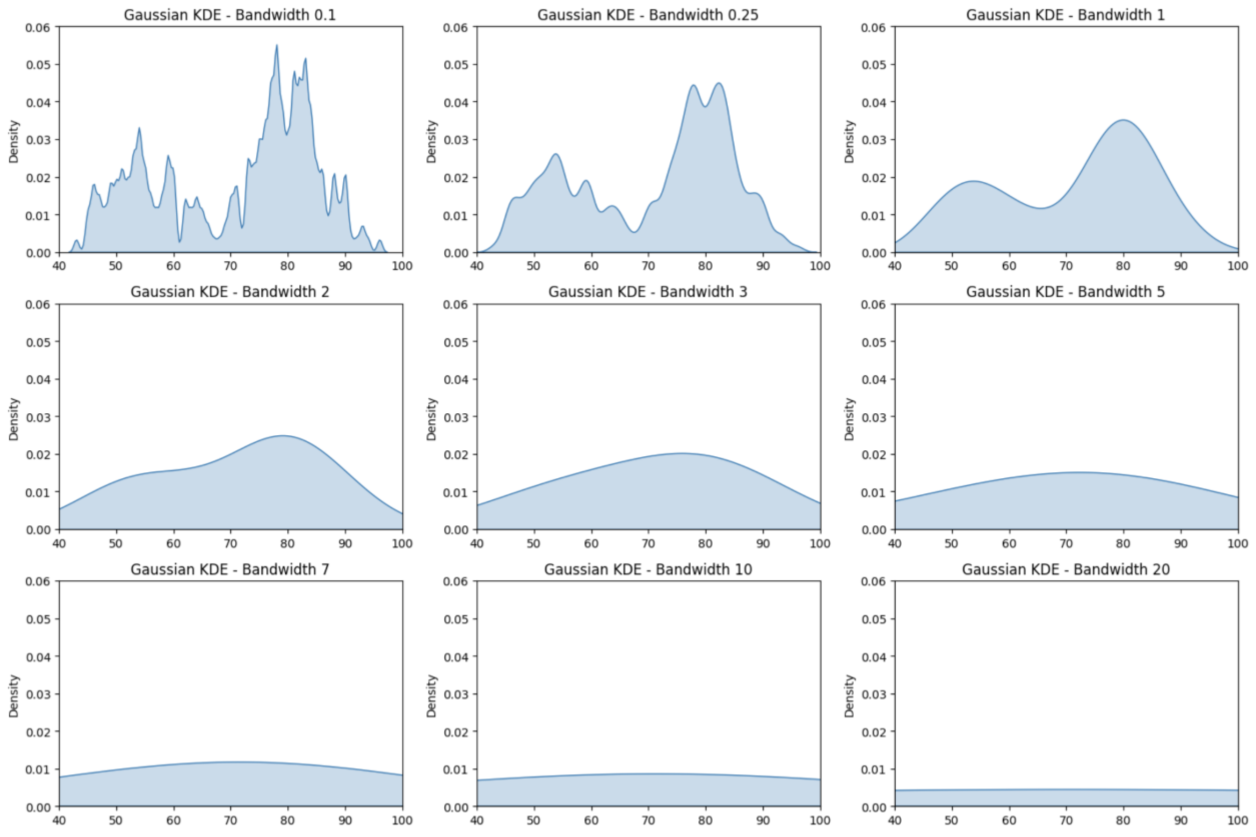
A kernel is usually considered to be a symmetric probability density function (pdf):

- $K_h(u) \geq 0$ (non-negative)
- $\int K_h(u)du = 1$ (integrates to one)
- $K_h(u) = K_h(-u)$ (symmetric about 0)
- Notice that if the kernel has compact support, so does the resulting density estimate

- The Gaussian kernel is the most popular, but has infinite support
    - This is good when the true density has infinite support
    - However, it requires more computation than kernels with finite support
    - It is familiar and properties are well understood

## 4.3 Bandwidth

- The bandwidth parameter, $h$ controls the amount of smoothing
    - It is equivalent to the bin size parameter for histograms
- What happens to the density estimate when $h \uparrow \infty$?
- What happens to the density estimate when $h \downarrow 0$?
- **The choice of bandwidth is much more important than the choice of kernel**
- Bandwidth is usually defined as the standard deviation of the kernel
    - But not always, so check the implementation.



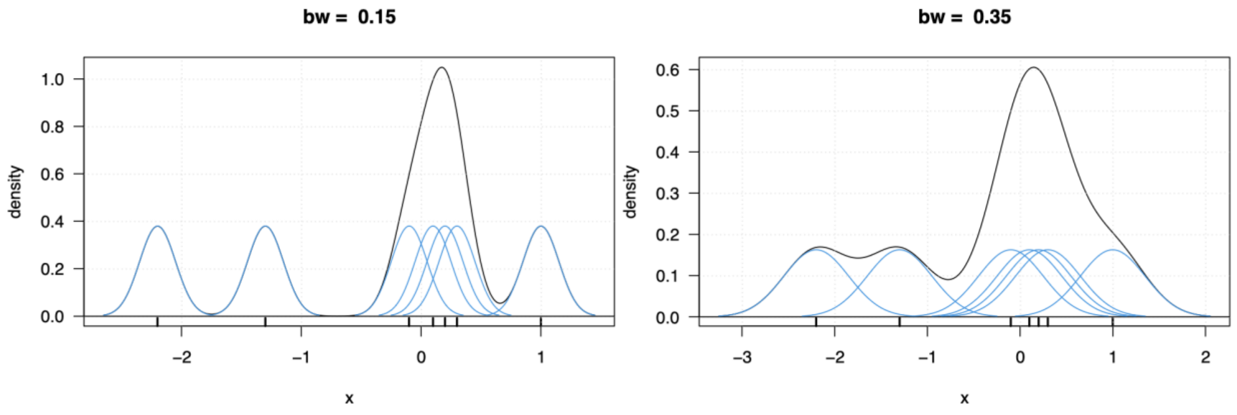
### 4.4 KDE Demo

Check out the shiny app for visualizing the effects of the smoothing parameters on density histograms and kernel density estimation.

### 4.5 Another Perspective

- We have described KDE as taking a local density around location *x*
- An alternative perspective is to view KDE as an *n* component *mixture model* with mixture weights of 1/*n*.

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x_i - x) = \sum_{j=1}^{n} \frac{1}{n} f_i(x) \ \left(f_i(x) = K_h(x_i - x)\right)$$



### 4.6 Bandwidth Selection

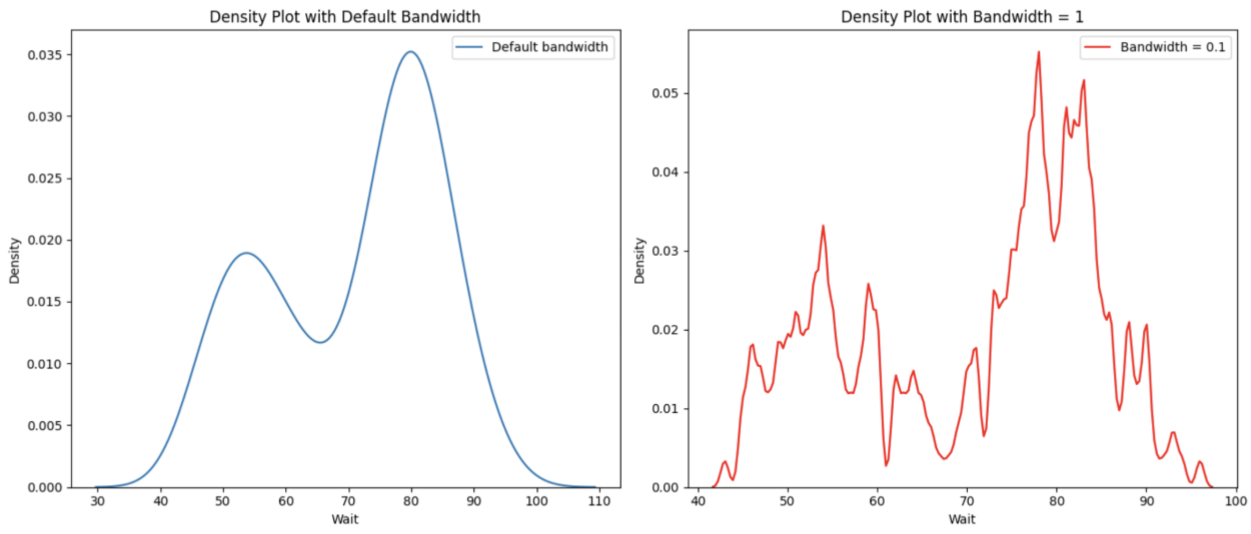
- The best bandwidth is the one that is best for your problem
  - E.g., *The best bandwidth for density estimation may not be best for classification*
  - Choosing a bandwidth that is visually appealing may be suitable
- For density estimation, we want a "Goldilocks" bandwidth; one that produces a density that is not *too wiggly* nor *too smooth*.
- Or to state it plainly, when the goal is density estimation, we want to find the bandwidth that gives a density estimate closet to the true density. But,
  - We don't know the true density
  - There are many ways to define "close"
- For this class, we will not go into the details other than to say most bandwidth selection methods are based on *plug-in* or *cross-validation.*

#### 4.6.1 Bandwidth Selection in Python

There are a few KDE functions in Python.

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Old Faithful data
faithful_dataset = sm.datasets.get_rdataset("faithful", "datasets")
faithful_data = faithful_dataset.data
wait = faithful_data['waiting'].values

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Subplot 1: Default bandwidth (smoother KDE)
sns.kdeplot(wait, ax=axes[0], label="Default bandwidth")
axes[0].set_title("Density Plot with Default Bandwidth")
axes[0].set_xlabel("Wait")
axes[0].set_ylabel("Density")
axes[0].legend()

# Subplot 2: Specified bandwidth of 1 (less smooth)
sns.kdeplot(wait, bw_adjust=0.1, color="red", ax=axes[1], label="Bandwidth =
0.1")
axes[1].set_title("Density Plot with Bandwidth = 1")
axes[1].set_xlabel("Wait")
axes[1].set_ylabel("Density")
axes[1].legend()

plt.tight_layout()
plt.show()
```

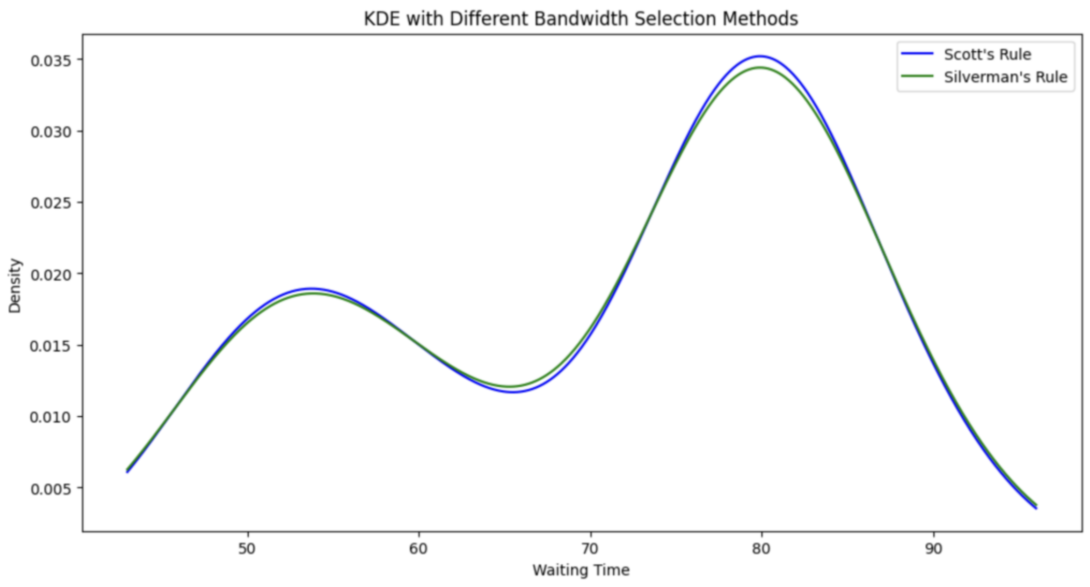


Bandwidth selection is a crucial step in KDE. Python libraries, primarily Scipy, Statsmodels, and Scikit-learn, provide methods to estimate the bandwidth. Some of the bandwidth selection methods available in these libraries are:

1. **Scipy**:
   - scipy.stats.gaussian_kde: This function uses Scott's Rule by default but also allows Silverman's Rule of Thumb. These can be specified using the bw_method argument.
2. **Other Libraries**:

- If you are specifically looking for methods such as plug-in (hpi), least squares cross-validation (hlscv), smoothed cross-validation (hscv), and unbiased cross-validation (hucv), they are not natively present in popular Python libraries as predefined functions. However, these methods are common in the R ecosystem, especially with the ks package. If you need to utilize these specific methods in Python, you might have to implement them manually or search for specialized libraries that might have implemented these.

For most common applications, the default methods provided by Scipy, Statsmodels, or Scikit-learn are sufficient.


KDE with Different Bandwidth Selection Methods
Scott's Rule
Silverman's Rule
Density
Waiting Time