

Probability Distribution Functions (VIMP) -PDF, PMF & CDF

Random Variables

What are **Random Variables** in Stats and Probability?

- A Random Variable is a set of possible values from a random experiment.
- A RV can hold any value.
 - eg. {H, T}, {1,2,3,4,5,6}
 - These outcomes are called **sample space**
- You denote RV with CAPITAL LETTERS
- Algebra variables → small letters

2 Types of RVs

1. Discrete RV

- eg. {1,2,3,4,5,6}
- No decimal values

2. Continuous RV

- Range of values
- eg. CGPA → from 1 to 10

Probability Distributions

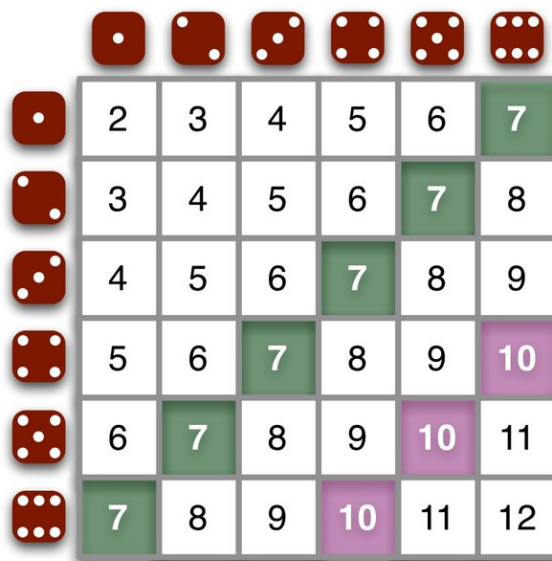
- A probability distribution is a list of all of the possible outcomes of a random variable along with their corresponding probability values.

<u>coin toss</u>	1 (H)	0 (T)
probab	$\frac{1}{2}$	$\frac{1}{2}$

- Possibilities in a dice roll are- $\{1,2,3,4,5,6\}$
 - The probability is $1/6$

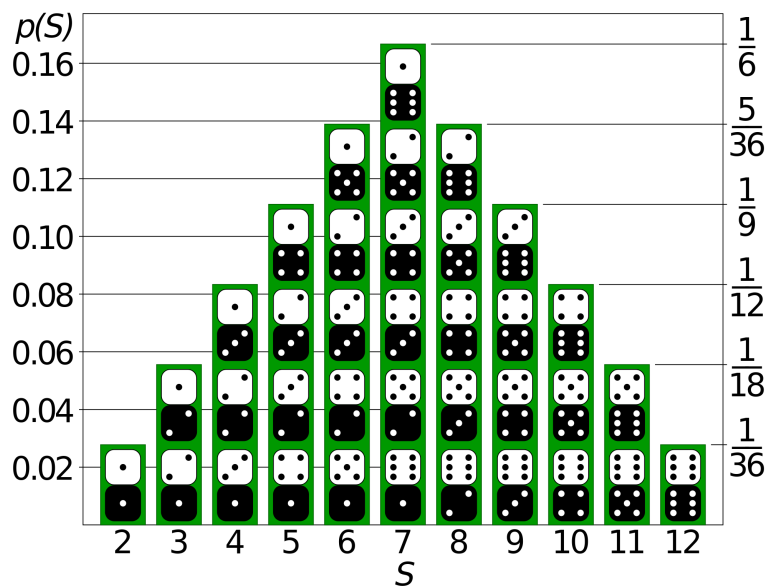
If we roll 2 dice

- We want probability of the sum



	2	3	4	5	6	7
3	3	4	5	6	7	8
4	4	5	6	7	8	9
5	5	6	7	8	9	10
6	6	7	8	9	10	11
7	7	8	9	10	11	12

Sum	Possible Cases	Favourable Cases	Probability
2	(1,1)	1	1/36
3	(1,2), (2,1)	2	2/36
4	(1,3), (2,2), (3,1)	3	3/36
5	(1,4), (2,3), (3,2), (4,1)	4	4/36
6	(1,5), (2,4), (3,3), (4,2), (5,1)	5	5/36
7	(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)	6	6/36
8	(2,6), (3,5), (4,4), (5,3), (6,2)	5	5/36
9	(3,6), (4,5), (5,4), (6,3)	4	4/36
10	(4,6), (5,5), (6,4)	3	3/36
11	(5,6), (6,5)	2	2/36
12	(6,6)	1	1/36
Total		36	

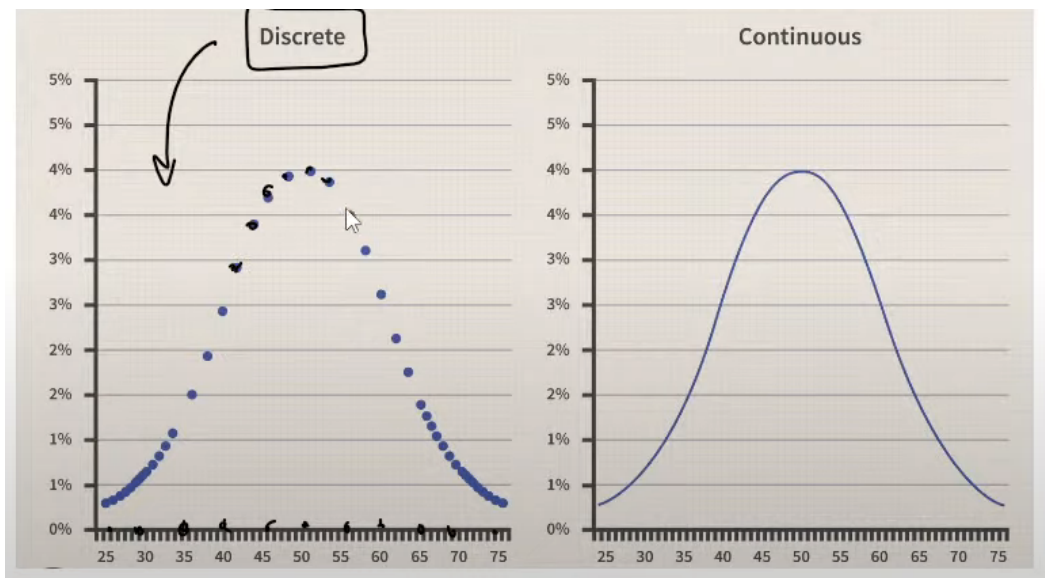
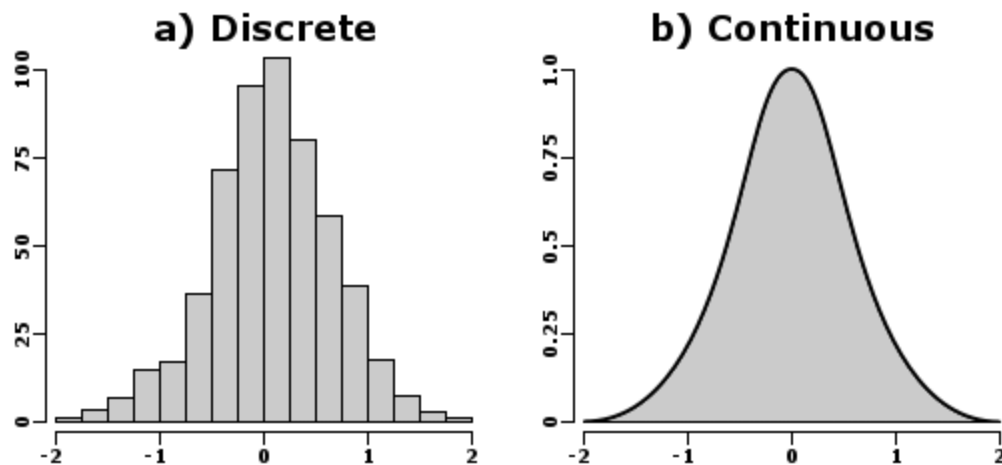


- The probability of 7 is highest
- The probability of 2 & 12 is lowest
- You can't make these tables for big values or cont. random variables like CGPA
- To solve this problem, we can make a function $\rightarrow y = f(x)$
- We can plot its graph

Note - A lot of time Probability Distribution and Probability Distribution Functions are used interchangeably.

Types of Probability Distributions

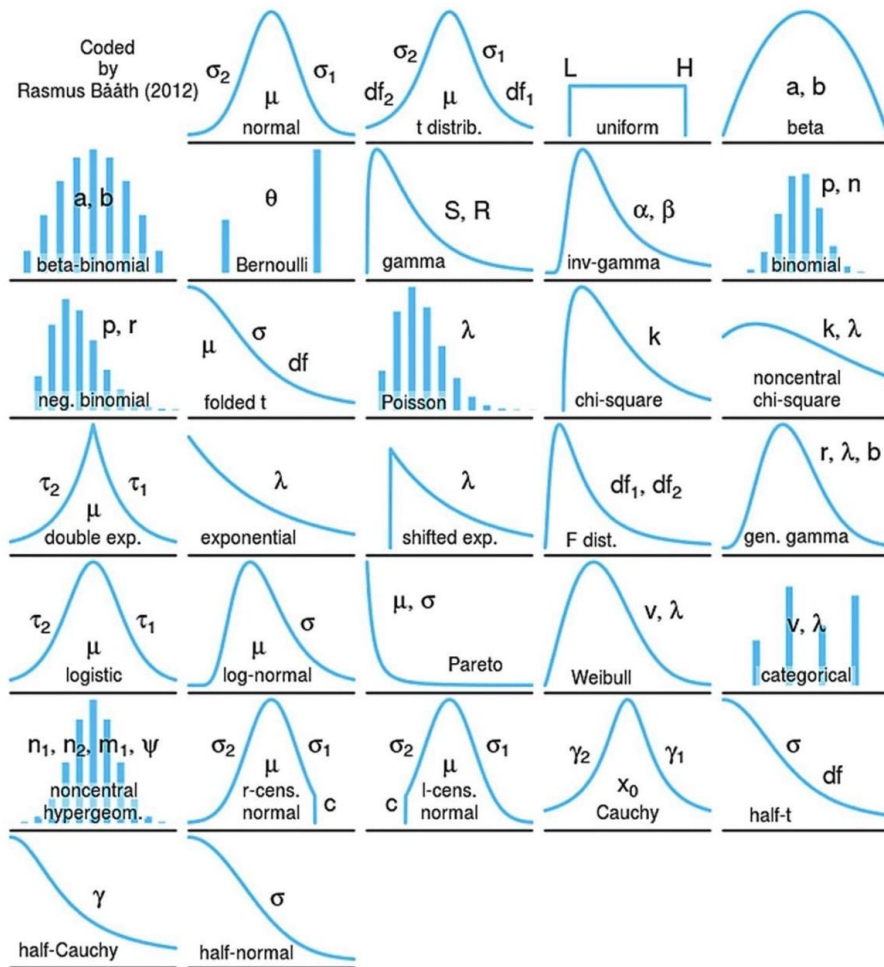
1. Discrete
2. Continuous

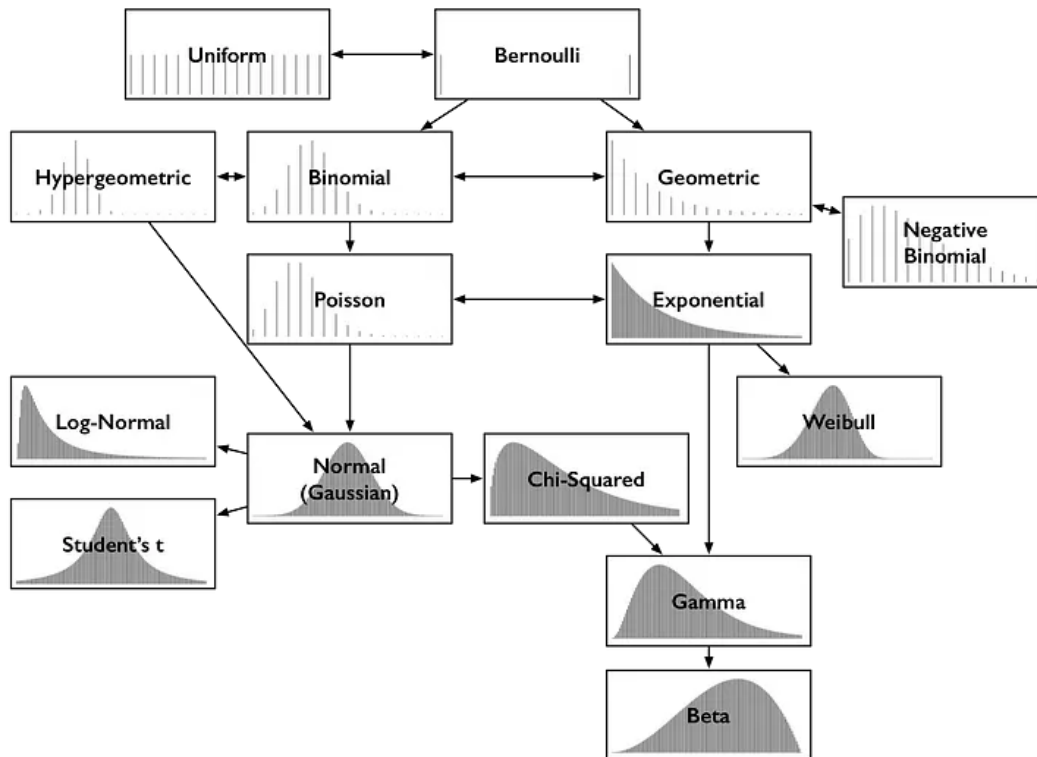


- We don't have all the values in discrete

Famous Probability Distributions

Probability Distributions





- If our data follows a famous distribution then we automatically know a lot about the data.

Parameters

- Parameters in probability distributions are numerical values that determine the **shape, location, and scale of the distribution**.
- Different probability distributions have different sets of parameters that determine their shape and characteristics, and understanding these parameters is essential in statistical analysis and inference.

Quick Summary of Some Common Parameters:

Parameter	Distribution	What it Controls
λ (lambda)	Poisson, Exponential	Rate of occurrence (how frequent events happen)
a, b	Uniform	The range of possible values (min and max)
θ (theta)	Gamma, Exponential, Beta	Shape and scale (how spread out the distribution is)

Parameter	Distribution	What it Controls
α (alpha)	Beta, Gamma	Shape (how the distribution behaves)
β (beta)	Beta, Gamma	Shape and scale (controls skew and spread)
μ (mu)	Normal	Mean or center of the distribution
σ (sigma)	Normal	Standard deviation (spread of values)
ρ (rho)	Bivariate (2 variables)	Correlation between two variables

Probability Distribution Functions

- Mathematical function that describes the probability of obtaining different values of a random variable in a particular probability distribution.

2 Types of Probability Distribution Functions

- Probability **Mass** Function (PMF)
 - Function for **Discrete Random Variable**
 - eg. dice roll
- Probability **Density** Function (PDF)
 - Function for **Continuous random variable**
 - eg. height, CGPA
- Cumulative Distribution Functions (CDF)**
 - Combination of the above 2

Probability **Mass** Function (PMF)

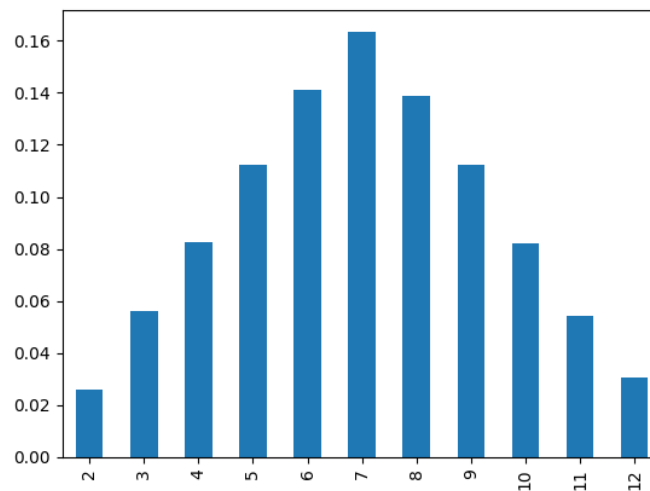
- Mathematical function that describes the probability distribution of a **discrete random variable**.
- It assigns a probability to each possible value of the random variable.
- The probabilities assigned by the PMF must satisfy two conditions:

- a. The probability assigned to each value must be non-negative.
- b. The sum of the probabilities assigned to all possible values must equal 1.

```
import pandas as pd
import random
l=[]
for i in range(10000):
    a= random.randint(1,6)
    b= random.randint(1,6)

    l.append(a+b)
```

```
t= (pd.Series(l).value_counts()/pd.Series(l).value_counts().sum()).sort_index()
t.plot(kind='bar')
```



2 Famous Distributions in PMF

1. Bernoulli distribution →
2. Binomial distribution →

Cumulative Distribution Functions (CDF)

- We're going to calculate CDF for PMF
- The **Cumulative Distribution Function (CDF)** tells you the **cumulative probability** that a random variable will take a value **less than or equal to** a certain number.

Ex.

Let's say you're rolling a

fair six-sided die. The CDF helps us find the probability that the die shows a value **less than or equal to** a given number.

- If you want to know the probability of rolling a 3 or less (that means 1, 2, or 3), you would calculate it using the CDF.

CDF Calculation:

1. For **$P(X \leq 1)$** : The probability that the die shows 1 or less is just $1/6$.
2. For **$P(X \leq 2)$** : The probability that the die shows 2 or less is $1/6 + 1/6 = 2/6$.
3. For **$P(X \leq 3)$** : The probability that the die shows 3 or less is $1/6 + 1/6 + 1/6 = 3/6$.
4. For **$P(X \leq 4)$** : The probability that the die shows 4 or less is $4/6$.
5. For **$P(X \leq 5)$** : The probability that the die shows 5 or less is $5/6$.
6. For **$P(X \leq 6)$** : The probability that the die shows 6 or less is $6/6 = 1$.

Outcome (x)	CDF $F(X \leq x)$
1	1/6
2	2/6
3	3/6
4	4/6
5	5/6
6	6/6 = 1

Key Points About CDF:

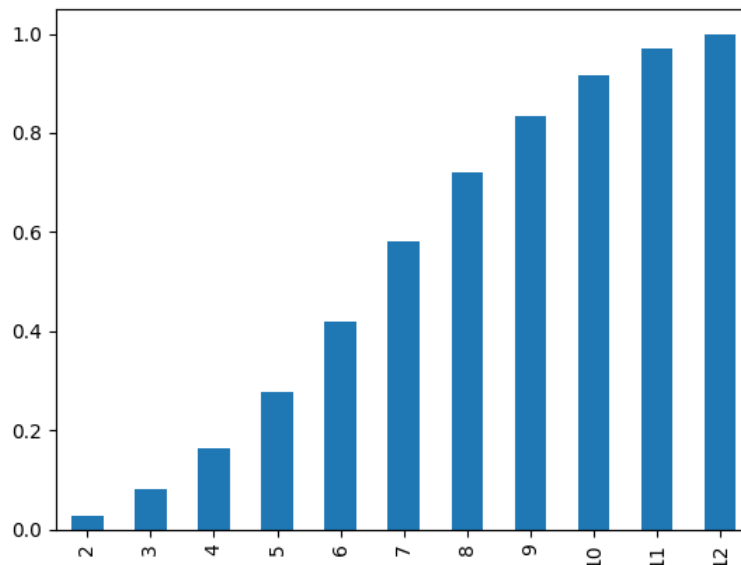
1. **CDF is non-decreasing:** The CDF will never decrease as x increases because you're always adding more probability as you move to higher values.
2. **CDF gives the cumulative probability:** It tells you the total probability that the random variable is less than or equal to a certain value.
3. **For Discrete Variables:** The CDF is calculated by summing the probabilities for each possible value up to the point you're interested in.
4. **Range of CDF:** The CDF always ranges from 0 to 1.
 - $P(X \leq x) = 0$ when the random variable is at its lowest possible value.
 - $P(X \leq x) = 1$ when the random variable is at its highest possible value (or goes to infinity for continuous variables).

- **For discrete variables:**

$$F(x) = P(X \leq x) = \sum_{i \leq x} P(X = i)$$

- Let's plot a graph for this

```
t.cumsum().plot(kind='bar')
```



Probability Density Function (PDF) (VVIMP-Difficult)

- **For continuous random variables**, you can't get an exact probability for a single point (since there are infinitely many possible values). Instead, the PDF gives you a way to calculate the **probability** that the random variable will fall within a **range of values**.
- **Continuous Variables**: PDFs are used for continuous variables, like **time**, **height**, **weight**, or **temperature**, where the variable can take any value within a certain range.
- **Area Under the Curve**: The area under the curve of the PDF represents the total probability for that range of values.
 - The **total** area under the curve from $-\infty$ to $+\infty$ is always **1** (since the total probability for all possible outcomes is 1).
- **Probability Between Two Values**: To find the probability that the random variable lies between two values, you need to calculate the **area** under the curve between those two points.



In PDF, on Y-axis, there is **Probability Density**, not probability.

- In PMF, Y- axis represents probability.

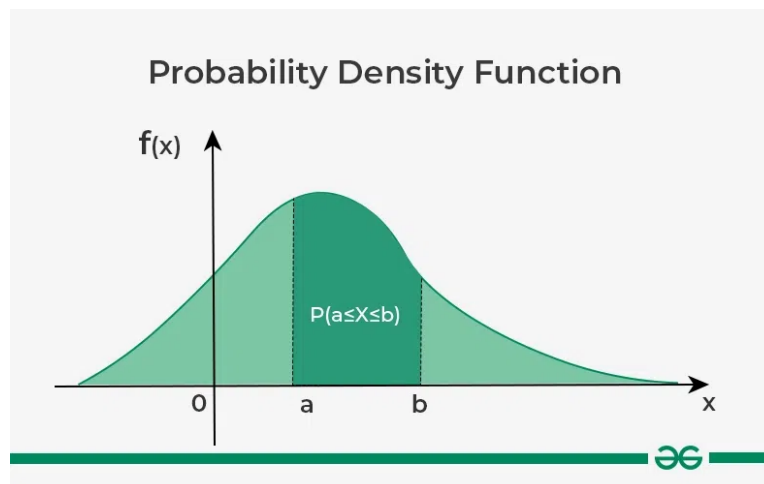
Why no probability on Y-axis?

- Since there are infinite values on X-axis, the probability of any point becomes close to zero.
- To calculate probability from point a to b → Calculate the **AUC** with the help of integration.



Probability Density Function (PDF) **cannot** give you the probability of a specific value.

Visualizing a PDF:



Let's say we have a **normal distribution** (a bell curve), which is one of the most common distributions. The **PDF** of a normal distribution is shaped like a bell curve.

- **High peak** near the mean: The value at the center (the mean) has the highest probability density.
- **Tail ends**: The probability density gets smaller as you move away from the mean.



The **PDF curve** doesn't give you the probability of a single point (since the area of a single point is 0), but rather, the **area under the curve** between two points gives the **probability** for that range.

Formula of PDF:

The PDF function is often denoted as **$f(x)$** , where **x** is the value of the random variable.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Where:

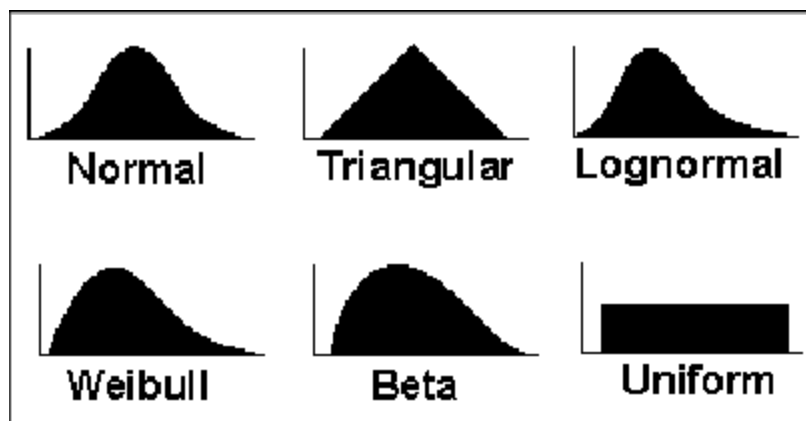
- **μ** is the mean (average) of the distribution.
- **σ** is the standard deviation, which measures the spread of the distribution.
- **x** is the value of the random variable.
- **e** is Euler's number (approximately 2.71828), which is used in exponential functions.

- **Probability for a Range:** The probability that a random variable **X** falls within a range $a \leq X \leq b$ is given by the **integral** of the PDF from **a** to **b**:

$$P(a \leq X \leq b) = \int_a^b f(x) dx$$

This is the area under the curve between **a** and **b**.

Famous probability density functions



Density Estimation

- Used to plot graph of PDF

2 Methods of density estimation:

1. Parametric

- assume that the data follows a **specific probability distribution** (such as a normal distribution)

2. Non-parametric

- Non-parametric methods **do not make any assumptions** about the distribution and instead estimate it directly from the data.
- Commonly used techniques for density estimation include kernel density estimation (**KDE**), **histogram** estimation, and **Gaussian mixture models (GMMs)**.

Parametric Density Estimation

- You observe the data
- If it seems like any a specific **probability distribution**, such as the normal, exponential, or Poisson distributions, you do the calculations as per **that** distribution.

```
from numpy.random import normal
sample = normal(loc=50, scale=5, size=1000)
```

```
array([49.15433102, 58.00238273, 50.8508338 , 52.94516038, 55.32744477,
       55.19828179, 47.02427827, 47.84524389, 43.49660123, 46.23461512,
       46.95823141, 47.98940944, 49.06915026, 50.56599216, 54.43764479,
       40.42764873, 41.8200636 , 49.74581738, 47.94754148, 46.3010518 ,
       55.34850589, 41.57480115, 52.40107403, 55.88035711, 49.93034041,
       50.64046373, 55.14027618, 52.84581358, 44.64549775, 48.5396043 ,
       45.22521415, 56.01583418, 54.86010223, 46.74656974, 46.33196532,
       52.77223092, 52.40169698, 51.83764445, 53.03663227, 58.01731527,
       49.48948414, 42.0243499 , 55.4321095 , 42.24936595, 59.61252281,
       50.12559916, 51.01831185, 55.57796219, 45.58209635, 40.46766924,
```

`normal()` → Gaussian normal distribution

`loc=50` : This parameter sets the **mean** (or expected value) of the normal distribution to 50.

- The average value of the generated numbers will tend to be around 50.

`scale=5` : This parameter sets the **standard deviation** of the distribution to 5.

`size=1000` : This parameter specifies the **number of random numbers** to generate.

```
sample.mean()
```

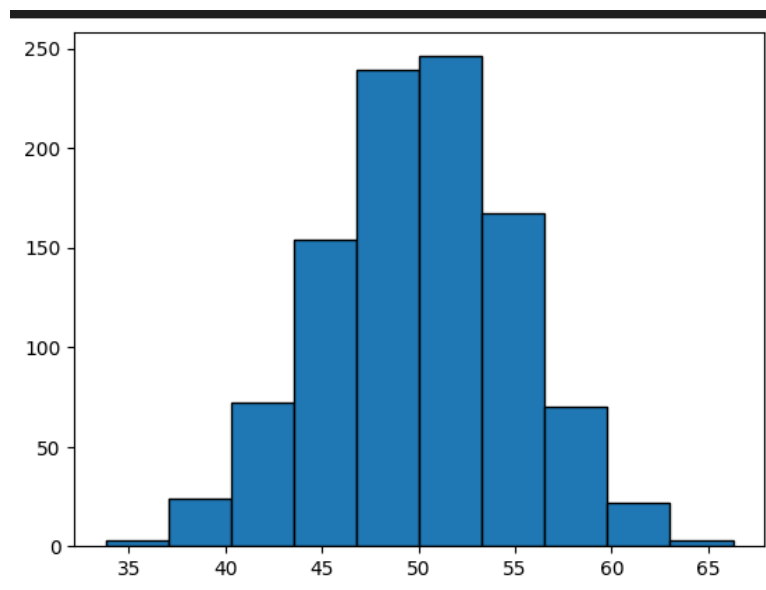
Output: 50.04290515347713

```
sample.std()
```

Output: 4.982279186251664

Plot histogram to understand the distribution of data

```
plt.hist(sample, bins=10, edgecolor='black')
```



- Store mean and SD in a variable


```
sample_mean = sample.mean()  
sample_std = sample.std()
```

Now, fit the distribution with the above parameters:

- μ = sample_mean &
- σ = sample_std

Put the above values in this formula:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- We will provide `x` & the formula will give us probability density

```
from scipy.stats import norm  
dist = norm(sample_mean, sample_std)
```

- The `norm` function provides tools for working with the normal (or Gaussian) probability distribution.
- This code defines a normal distribution with a mean and a standard deviation.
- This `dist` object can now be used to calculate probabilities, generate random samples, and perform other operations related to this specific normal distribution.
- The code says, "THIS IS STANDARD DISTRIBUTION."

- We will generate 100 data points between min & max values and send em to linspace.

```
values = np.linspace(sample.min(), sample.max(), 100)
```

```
array([33.80687445, 34.13481511, 34.46275578, 34.79069644, 35.11863711,
       35.44657778, 35.77451844, 36.10245911, 36.43039977, 36.75834044,
       37.08628111, 37.41422177, 37.74216244, 38.0701031 , 38.39804377,
       38.72598444, 39.0539251 , 39.38186577, 39.70980643, 40.0377471 ,
       40.36568777, 40.69362843, 41.0215691 , 41.34950976, 41.67745043,
       42.00539109, 42.33333176, 42.66127243, 42.98921309, 43.31715376,
       43.64509442, 43.97303509, 44.30097576, 44.62891642, 44.95685709,
       45.28479775, 45.61273842, 45.94067909, 46.26861975, 46.59656042,
       46.92450108, 47.25244175, 47.58038242, 47.90832308, 48.23626375,
       48.56420441, 48.89214508, 49.22008575, 49.54802641, 49.87596708,
       50.20390774, 50.53184841, 50.85978907, 51.18772974, 51.51567041,
       51.84361107, 52.17155174, 52.4994924 , 52.82743307, 53.15537374,
       53.4833144 , 53.81125507, 54.13919573, 54.4671364 , 54.79507707,
       55.12301773, 55.4509584 , 55.77889906, 56.10683973, 56.4347804 ,
       56.76272106, 57.09066173, 57.41860239, 57.74654306, 58.07448373,
       58.40242439, 58.73036506, 59.05830572, 59.38624639, 59.71418705,
       60.04212772, 60.37006839, 60.69800905, 61.02594972, 61.35389038,
       61.68183105, 62.00977172, 62.33771238, 62.66565305, 62.99359371,
       63.32153438, 63.64947505, 63.97741571, 64.30535638, 64.63329704,
       64.96123771, 65.28917838, 65.61711904, 65.94505971, 66.27300037])
```

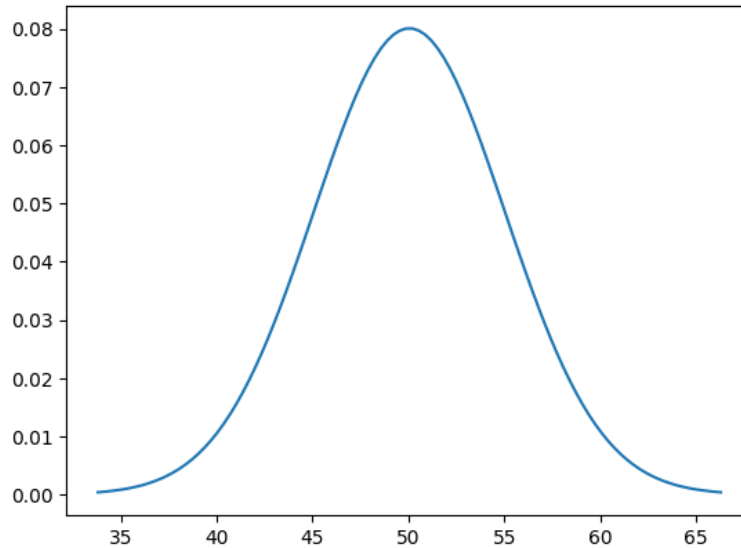
- **100 evenly spaced numbers** between min & max

`np.linspace()` is useful when you need a set of **evenly spaced numbers** over a specific range. For example:

- Plotting a graph with evenly spaced x-values.
- Generating test data for simulations or mathematical functions.

If we plot a graph of these values, it will look like this:

```
plt.plot(values, dist.pdf(values))
```



- `values` will represent the x-axis values (the points where you want to evaluate the PDF).
- `dist.pdf(values)` will be the y-axis values (the actual values of the PDF at the corresponding x points).
- `dist` : a **probability distribution object**
- `dist.pdf(values)` computes the **Probability Density Function** at each value in the `values` array.

PDF: The PDF represents the likelihood of a random variable taking a value in a continuous range. For a given point in the range, `pdf(x)` returns the probability density at that point.

```
probabilities = [dist.pdf(value) for value in values]
```

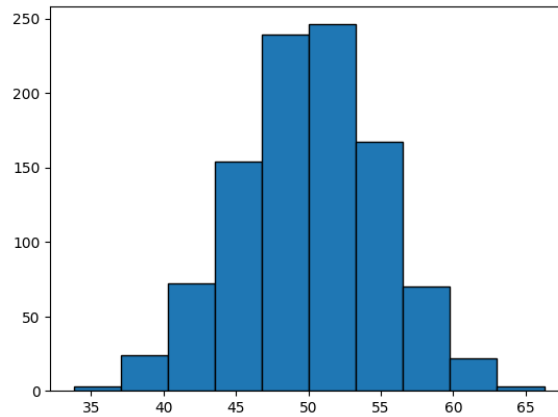
essentially the same thing as using

```
dist.pdf(values)
```

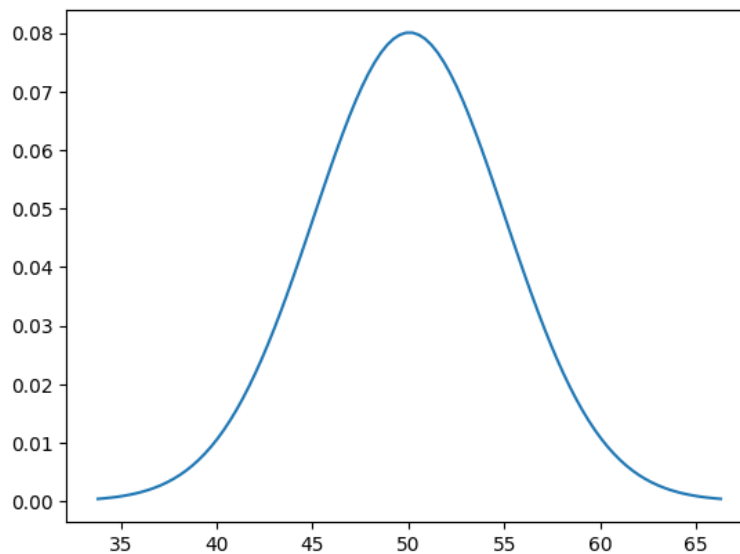
```
[0.0003958123392221197,
 0.0004894439203109526,
 0.0006026081272432787,
 0.0007387295439879269,
 0.0009016840525509817,
 0.0010958263681772687,
 0.0013260122652087568,
 0.0015976136889853099,
 0.0019165247940957258,
 0.0022891568354433856,
 0.0027224197795796858,
 0.003223688513428532,
 0.003800751619304281,
 0.004461740871240905,
 0.005215039898259924,
 0.006069170862476435,
 0.00703265851713115,
 0.008113871640204624,
 0.00932084257613216,
 0.010661066118066681]
```

- This calculates the **probabilities** of all the points in `values`
- The result is a list of PDF values, but it **loops through each element** of `values` and calls `dist.pdf()` on each individual element.
- This is **less efficient** than `dist.pdf(values)` because it's essentially doing the same calculation in a loop, which might be slower if `values` is large.
- The probability for each point ll be close to zero as mentioned above.
- Now, we can plot a graph for this

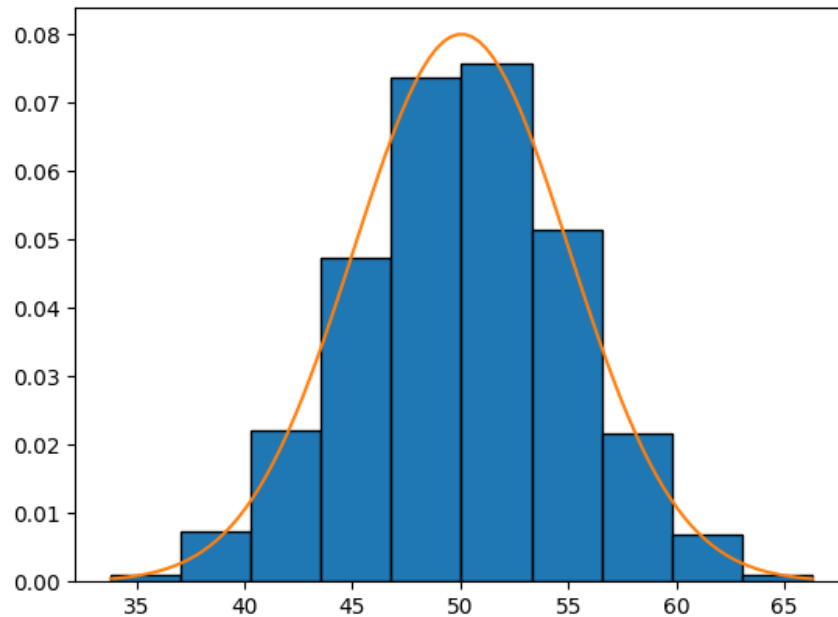
```
plt.hist(sample,bins=10, edgecolor='black');
```



```
plt.plot(values,probabilities)
```



```
plt.hist(sample,bins=10, edgecolor='black', density=True)  
plt.plot(values,probabilities)
```



Calculate the probability between 2 points:

a = 45

b = 55

```
# Calculate the probability between a and b
probability_between_a_and_b = dist.cdf(b) - dist.cdf(a)
probability_between_a_and_b
```

Output: 0.6843897549625151

- `dist.cdf(b)` gives the cumulative probability up to `b`.
- `dist.cdf(a)` gives the cumulative probability up to `a`.
- Subtracting these two values gives the **area under the curve** (or the probability) between `a` and `b`.

Non-Parametric Density Estimation (KDE)

- Sometimes the distribution is not clear or it's not one of the famous distributions.

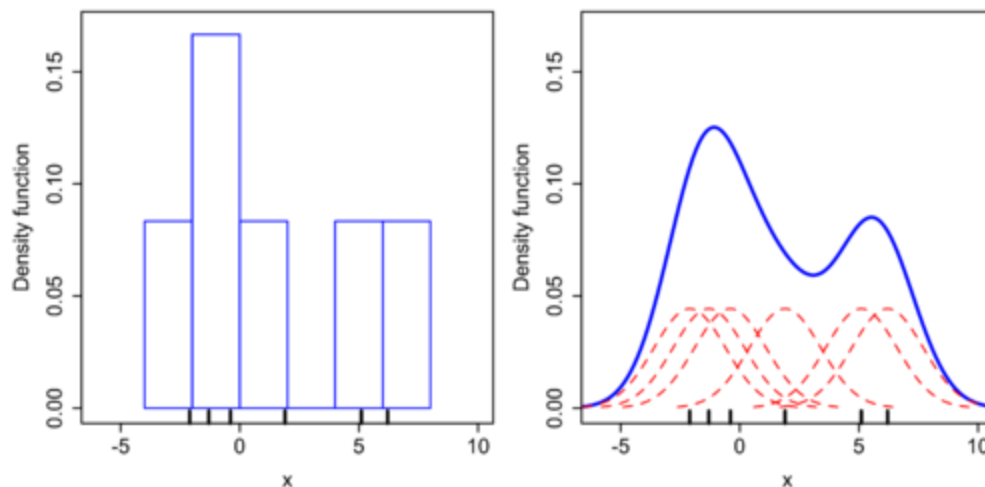
- Non-parametric density estimation is a statistical technique used to estimate the probability density function of a random variable without making any assumptions about the underlying distribution.
- This is typically done by creating a kernel density estimate

Advantages: it does not require the assumption of a specific distribution

Disadvantage: computationally intensive and may **require more data to achieve accurate estimates**

Kernel Density Estimate(KDE) (IMP -interviews)

- Technique involves using a kernel function to smooth out the data and create a continuous estimate of the underlying density function.
- **kernel** is a probability distribution- Generally it's gaussian



- You take a data point and generate a **normal distribution** around that data point
 - BUT you need SD for this → called as Bandwidth
 - You can adjust the Bandwidth
 - Less SD/ bandwidth = pointed graph
 - More SD = Smooth curve

- You do this with all the data points
- & you add them all

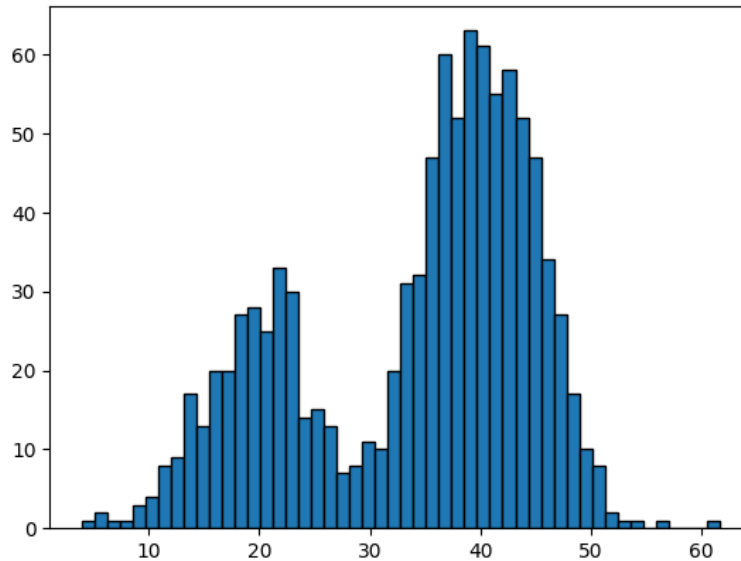
```
# generate a sample
sample1 = normal(loc=20, scale=5, size=300)
sample2 = normal(loc=40, scale=5, size=700)
sample = np.hstack((sample1, sample2))
```

```
array([11.82087684, 27.22220341, 22.86931028, 15.51731569, 18.52529025,
       12.23389654, 25.9797469 , 26.72258554, 13.54710533, 21.51035062,
       22.77352436, 21.93831083, 25.36463571, 12.58937165, 24.89926505,
        9.12991628, 11.64045208, 19.14867742, 25.9872177 , 35.28419731,
       15.213582 , 21.10679647, 21.86653856, 22.26876675, 14.84897925,
       17.00093715, 15.71651486, 14.81991811, 13.97320678, 19.16054092,
       23.24395941, 24.19330026, 24.22886415, 20.67332345, 27.13867497,
       23.9138589 , 21.65745501, 18.21619757, 24.51228917, 19.01446874,
       17.49714973, 16.36369321, 19.38628867, 21.46670693, 20.11478508,
       18.32969848, 19.282493 , 23.55182518, 24.72670659, 23.83163377,
       22.40673286, 25.75630711, 19.87187143, 22.45351409, 21.703179 ,
       25.04806102, 13.65170094, 21.80058888, 18.62120831, 28.44470226,
       24.987437 , 15.37671528, 6.23931186, 19.17779696, 14.83483793,
       12.74781241, 22.92241407, 18.934882 , 21.63976331, 24.12122906,
       15.87866433, 23.43955806, 14.145206 , 22.66418672, 21.82462765,
       22.9902081 , 23.80003792, 23.38778067, 13.41095652, 16.40339677,
       22.61846611, 22.12166529, 22.77348 , 17.78360586, 21.33188234,
       20.83351206, 20.77900625, 21.67011774, 18.70726692, 17.29490541,
       19.17236137, 18.43111309, 16.72746045, 24.7701785 , 22.01951476,
       21.76551997, 26.18163882, 24.15472961, 18.91597678, 22.01602626,
       22.44089963, 16.37324969, 23.04496909, 10.80313495, 19.35475532,
       20.02044333, 3.96601648, 19.95279974, 21.75874278, 25.06091357,
       10.55306655, 11.40000506, 26.34000517, 20.40000517, 23.44000517])
```

- 1000 points

We'll plot a histogram

```
plt.hist(sample, bins=50, edgecolor='black');
```

- This isn't following a particular distribution
- We will calculate the PDF of this using KDE.
- For that we have to make a Kernel object like we did with the normal one.
 - & we do it with `sklearn` library's `neighbours` package
 - We have to provide `bandwidth` (SD) & `kernel` (eg. gaussian)
- **& CONVERT YOUR DATA FROM 1D → 2D**
 - `.reshape(len(sample),1)`

install scikit-learn:

```
pip install scikit-learn
```

```
from sklearn.neighbors import KernelDensity  
model= KernelDensity(kernel='gaussian', bandwidth=3)
```

- we created an instance of the `KernelDensity` (a tool) that you can use to analyze your data

```
# convert data to a 2D array
sample = sample.reshape((len(sample), 1))

model.fit(sample)
```

- Fit the KDE: The KDE machine learns the pattern in your data by "fitting" it.
 - Think of it as "training" the KDE machine to understand your data.
 - When you run `.fit(data)`, the KDE machine looks at your data and learns how the data is distributed. It memorizes the positions of your data points so it can create a smooth curve around them.
- Now generate evenly spaced values from min to max.

```
x_vals = np.linspace(min(sample), max(sample), 100)
x_vals = x_vals.reshape((len(x_vals), 1))
```

- generate 100 numbers and reshape them → `reshape((len(x_vals), 1))`

Now, calculate the probabilities of all the x_values:

```
probabilities= model.score_samples(x_vals)
```

- The `score_samples` method in the `KernelDensity` class from scikit-learn is used to compute the **log of the estimated density** at specific data points.
- This gives us LOG values
- We have to convert these log values to normal values with `np.exp()`

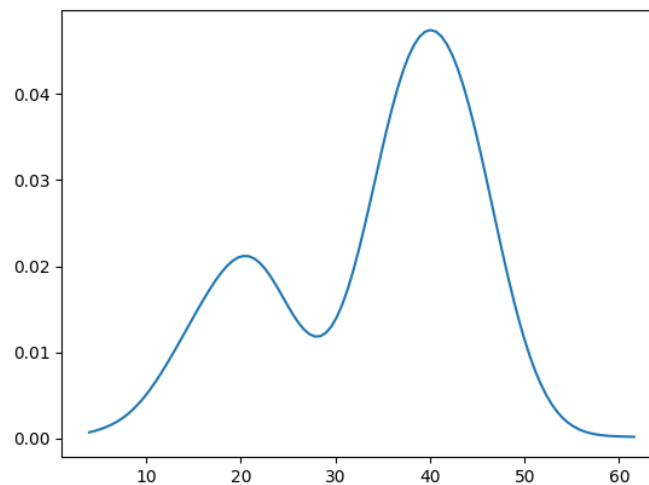
```
probabilities= np.exp(probabilities)
```

```
array([0.0006902 , 0.00085476, 0.00105151, 0.00128771, 0.00157206,  
       0.00191425, 0.00232423, 0.0028111 , 0.00338204, 0.00404127,  
       0.00478931, 0.00562275, 0.00653461, 0.00751504, 0.0085524 ,  
       0.00963435, 0.01074863, 0.01188346, 0.01302735, 0.01416843,  
       0.0152935 , 0.01638699, 0.01743006, 0.01840026, 0.0192716 ,  
       0.02001546, 0.02060211, 0.02100295, 0.02119325, 0.02115519,  
       0.02088086, 0.02037473, 0.01965525, 0.01875511, 0.01772006,  
       0.01660644, 0.01547759, 0.01439988, 0.01343882, 0.01265569,  
       0.01210506, 0.01183326, 0.01187742, 0.01226501, 0.01301341,
```

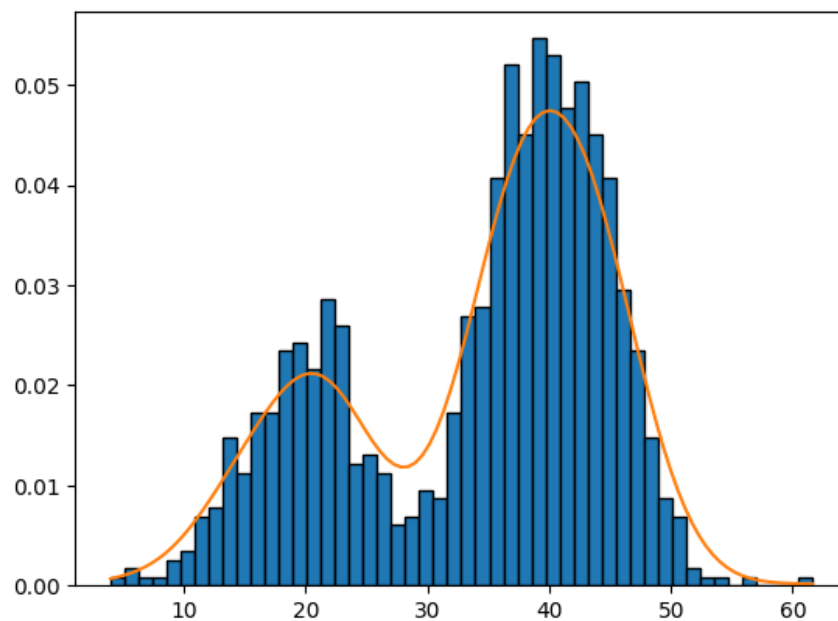
This gives us the probability of each point on X-axis

Now, plot the graph:

```
plt.plot(x_vals, probabilities)
```

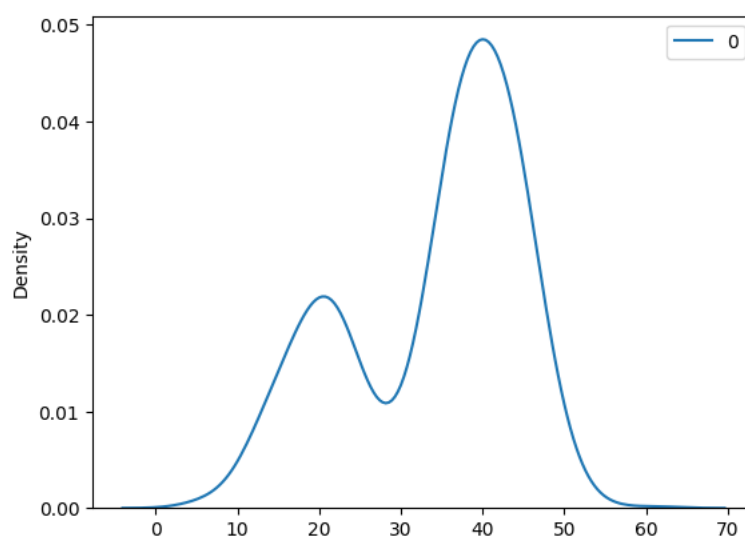


```
plt.hist(sample, bins=50, density=True, edgecolor='black')  
plt.plot(x_vals, probabilities)
```



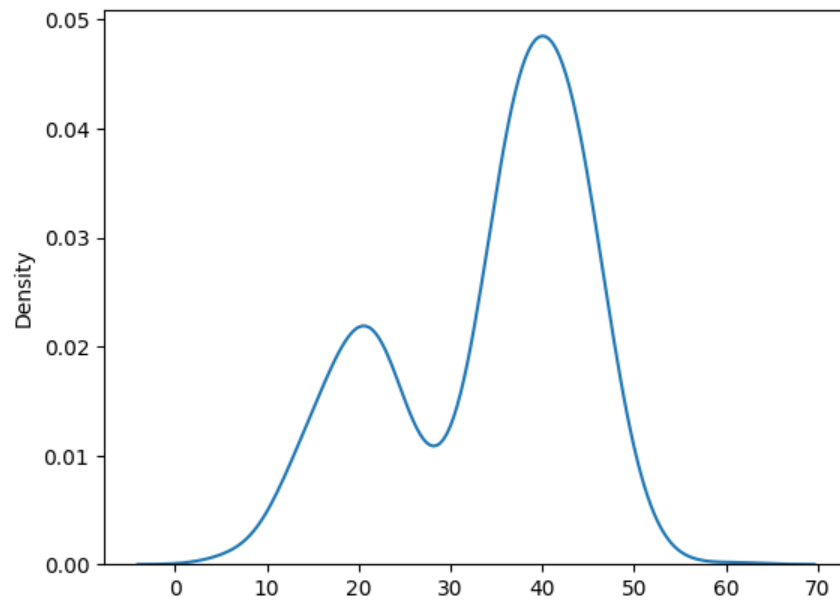
We can plot the same plot with seaborn

```
import seaborn as sns
sns.kdeplot(sample)
```



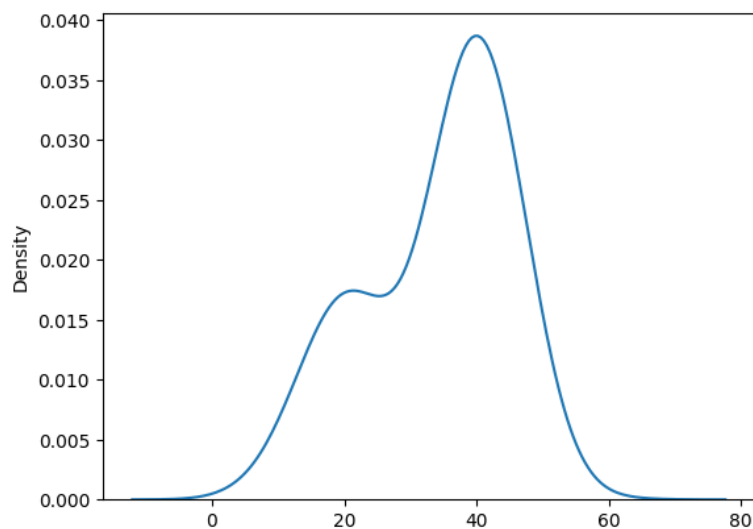
we can reshape the data

```
import seaborn as sns
sns.kdeplot(sample.reshape(1000));
```

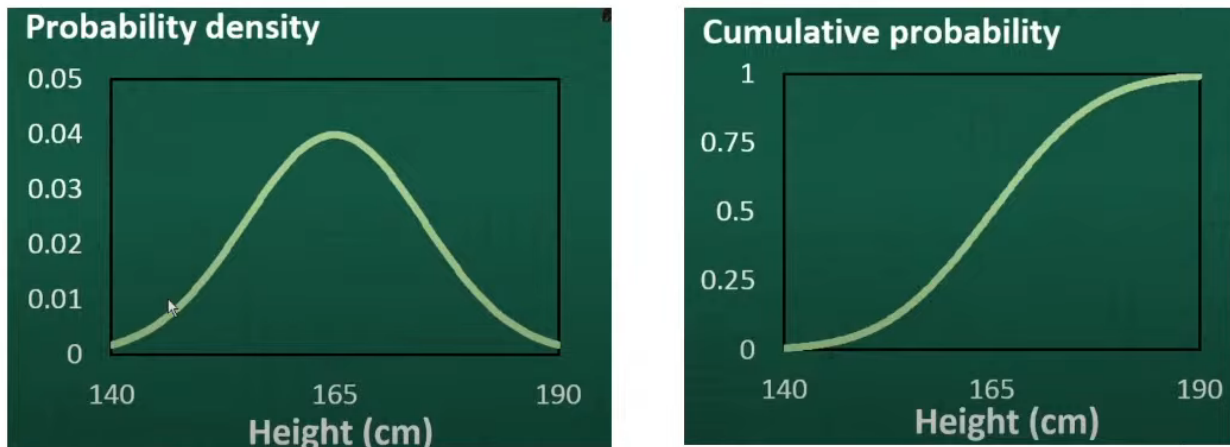


- Adjust bandwidth with `bw_adjust=`

```
import seaborn as sns
sns.kdeplot(sample.reshape(1000), bw_adjust=2);
```



Cumulative Distribution Function(CDF) of PDF



- If we perform **integration** on the probability density graph, we get the cumulative probability graph.
 - Here, we calculate AUC
- & if we perform **differentiation** on the cumulative probability graph, we get probability density graph.
 - Here, we calculate the slope at every point

How to use PDF in Data Science

- We'll use the famous *iris* dataset

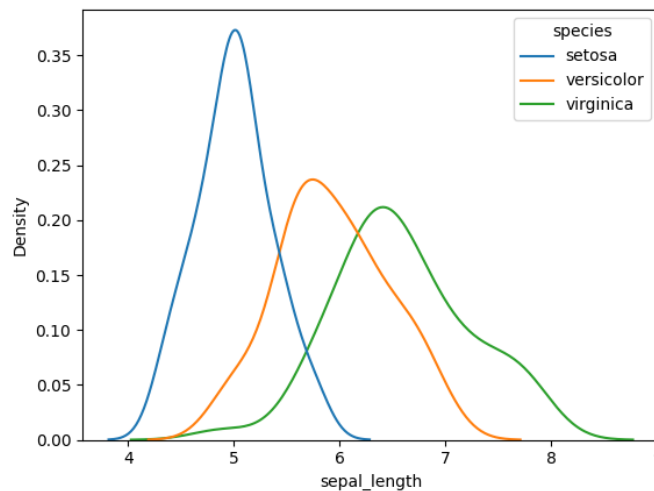
```
import seaborn as sns
df = sns.load_dataset('iris')
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

- We have to find out the type of flower based on the above information
- In these kinds of scenarios, you do feature selection.
 - You only keep the useful features
 - To find that out

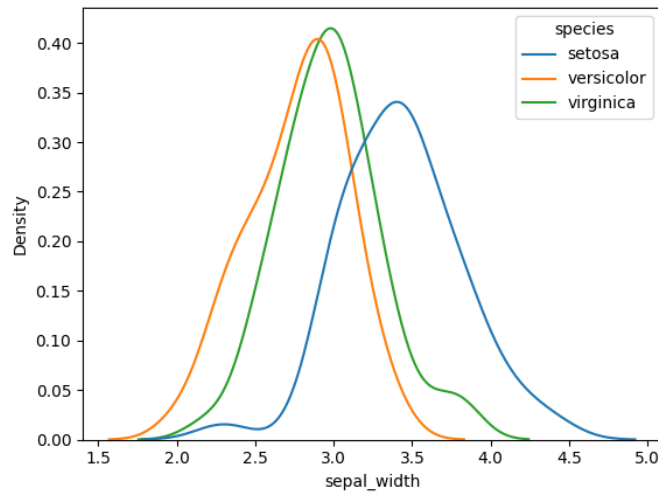
Let's compare Sepal_length

```
sns.kdeplot(data=df, x='sepal_length', hue='species')
```

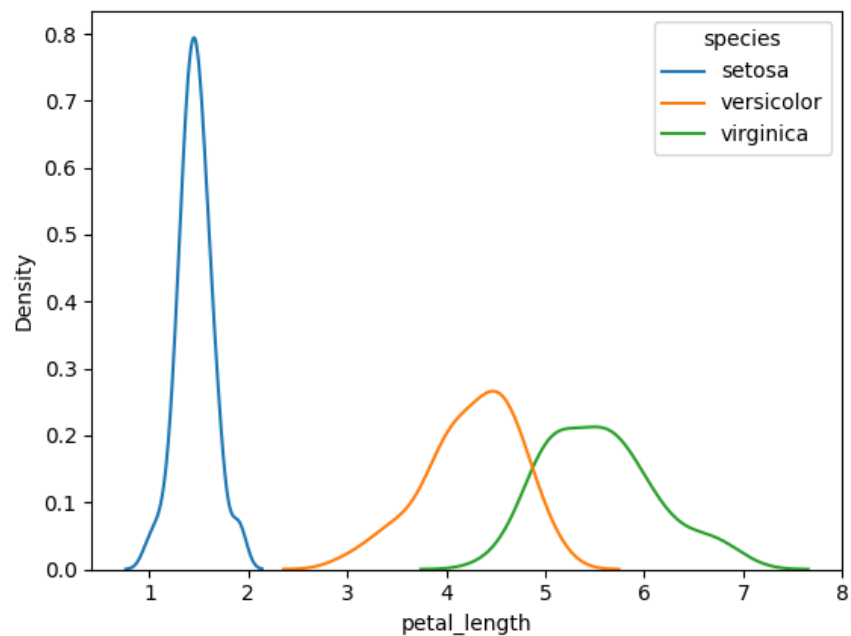


- Now repeat this code for remaining 2 parameters

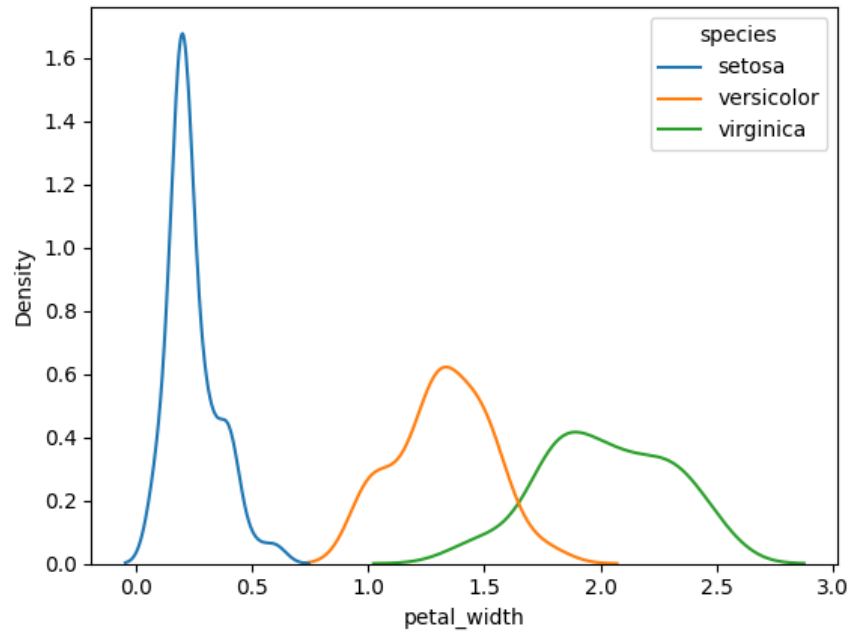
```
sns.kdeplot(data=df, x='sepal_width', hue='species')
```



```
sns.kdeplot(data=df, x='petal_length', hue='species')
```



```
sns.kdeplot(data=df, x='petal_width', hue='species')
```

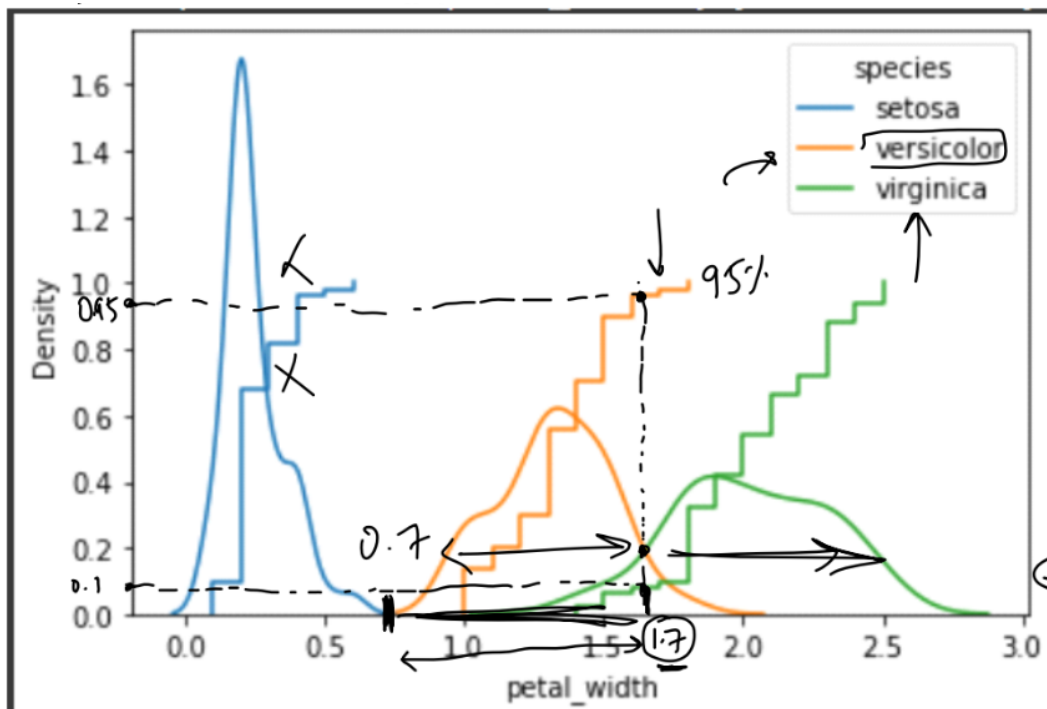
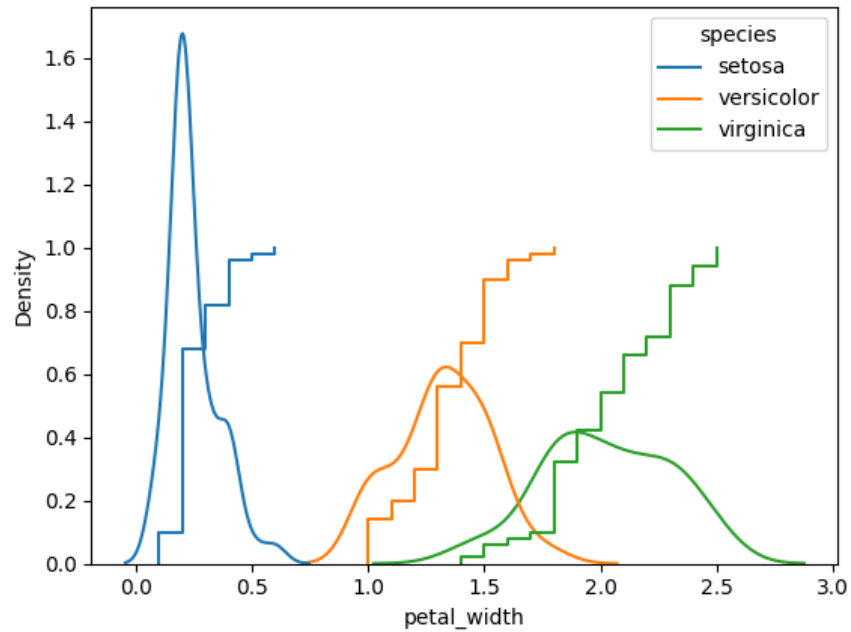



- In petal_width and petal_length graphs, we can separate the types clearly.
 - eg. In Petal_length → if <2.3, the flower is setosa
- In sepal columns, the graphs are overlapping, so it's difficult to make any separation

Calculate CDF for petal_width

`ecdfplot`

```
sns.kdeplot(df, x='petal_width', hue='species')  
sns.ecdfplot(df, x='petal_width', hue='species')
```

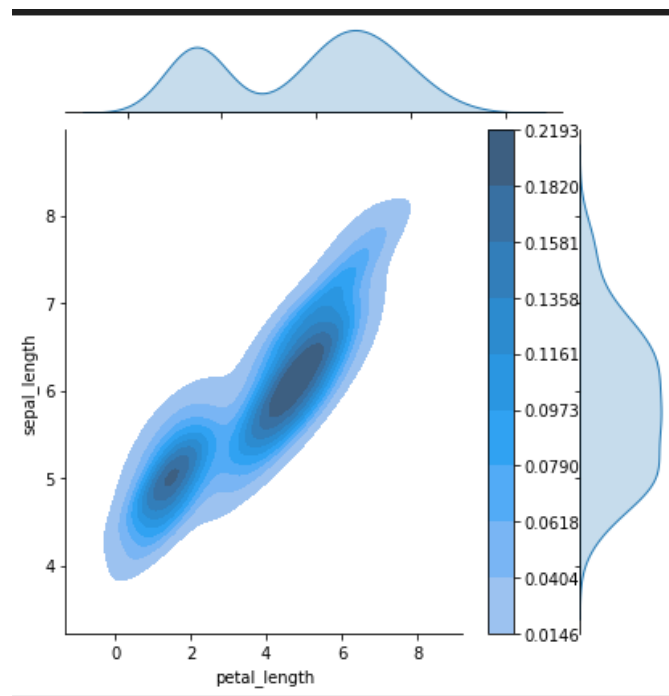


- From the above image, we can say that at the intersection point (1.7) → the petal width of 95% versicolor flowers is less than 1.7
 - Because the orange line intersects at 0.95

- **You'll be correct 95% of time**
- And, the petal width of the petal width of 10% virginica flowers is less than 1.7.
 - Because the green line intersects at 0.1
 - **You'll be correct 90% of time**

2D Density Plots

```
sns.jointplot(data=df, x="petal_length", y="sepal_length", kind="kde", fill=True, cbar=True)
```

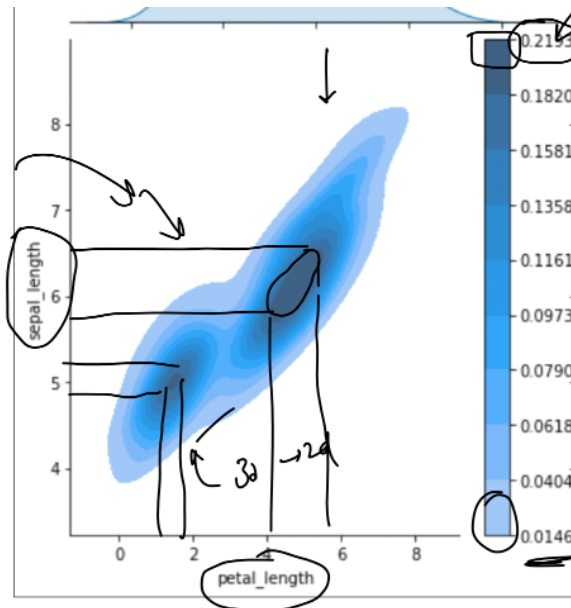


fill=True :

- When **fill=True**, the areas under the KDE curves will be filled with color. This is the shaded area beneath the density plot. If you set it to **fill=False**, you would only see the contour lines (no filling under the curve).

cbar=True:

- When `cbar=True`, a color bar will be added to the plot. The color bar shows the density values associated with the shading in the KDE plot, helping to visually interpret the density of the data points.



- These are high density areas
- **It shows which combinations' probability is higher.**
- Same can be visualised with line charts:

```
sns.lineplot(data=df, x='petal_length', y='sepal_length')
```

