Faculté des Sciences
كلية العلوم

Département de Physique

Master spécialise Master Intelligence Artificielle et Réalité Virtuelle

Project Report :

Realised by:

Zeghli Fatima Zahra

AIT OUAMER Ouafa

BOURASS Wiame

Supervised by:

Pr. Kaicer mohammed

Titled:

«Application of probability concepts on three different Datasets»

Année Universitaire : 2022/2023

I. Introduction

1. The Markov inequality and Bienaymé-Tchebychev inequality are used in probability theory and statistics to bound the probability of certain events.

2. The conditional expectation, also known as the expected value of a given event, is a fundamental concept in probability theory. It is a measure of the expected value of a random variable, given some condition or event. For example, the expected value of the roll of a dice, given that the roll is a 6, would be 6.

3. The Parzen window approach is a method for estimating the probability density function of a random variable from a set of observations. It involves choosing a window function and sliding it over the data, calculating the sum of the window function at each point and dividing by the total number of observations. The approach can be applied in two or three dimensions, depending on the number of variables being considered.

4. Point estimation, confidence interval estimation, and maximum likelihood estimation are all methods used in statistical inference to estimate the value of a population parameter based on sample data. Point estimation involves estimating the parameter with a single value, while confidence interval estimation gives a range of possible values for the parameter. Maximum likelihood estimation involves finding the parameter value that maximizes the likelihood function, which is a measure of the probability of the observed data given the parameter value.

II. Bounding a probability

1. Markov's Inequality :

The Markov bound is typically expressed as a function of the form:

__Figure 1: Markov's Inequality__

$$P(X \geq a) \leq \frac{E[X]}{a}$$

proof

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

$$\int_a^\infty \frac{x}{a} f(x)dx \leq \qquad \text{because } x > a \text{ for all } x \in (a, \infty)$$

$$\frac{1}{a}\int_0^\infty xf(x)dx = \qquad \text{because} \int_0^a xf(x)dx \geq 0$$

$$\frac{1}{a}\int_{-\infty}^\infty xf(x)dx = \qquad \text{because } f(x) = 0 \text{ for all } x < 0$$

$$\frac{E[X]}{a}$$

Where

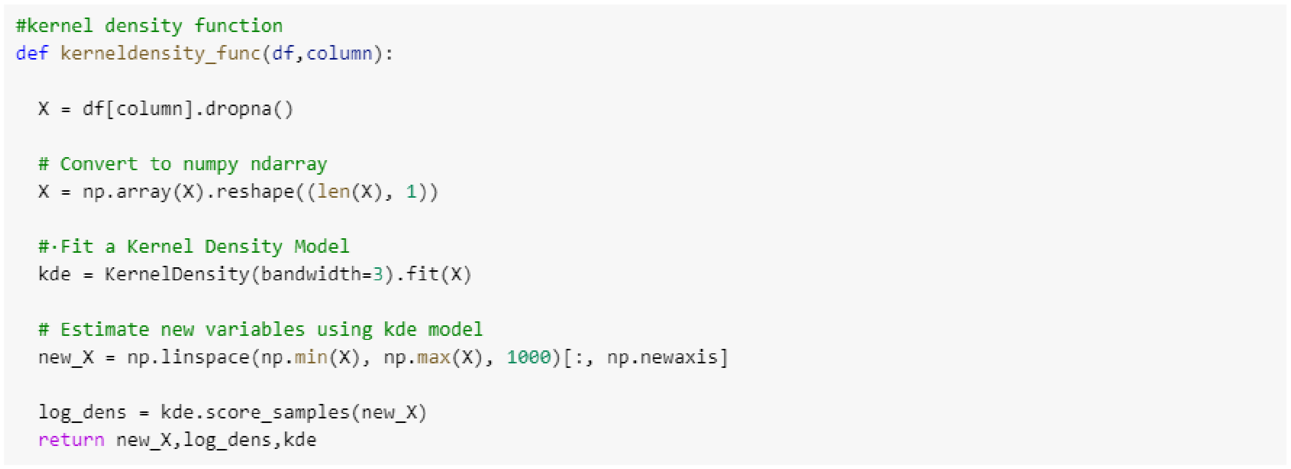
i. P(X >= k) is the probability that the event X will occur at least k times

ii. E[X] is the expected value of X

iii. k is a positive integer parameter that represents the number of times the event must occur

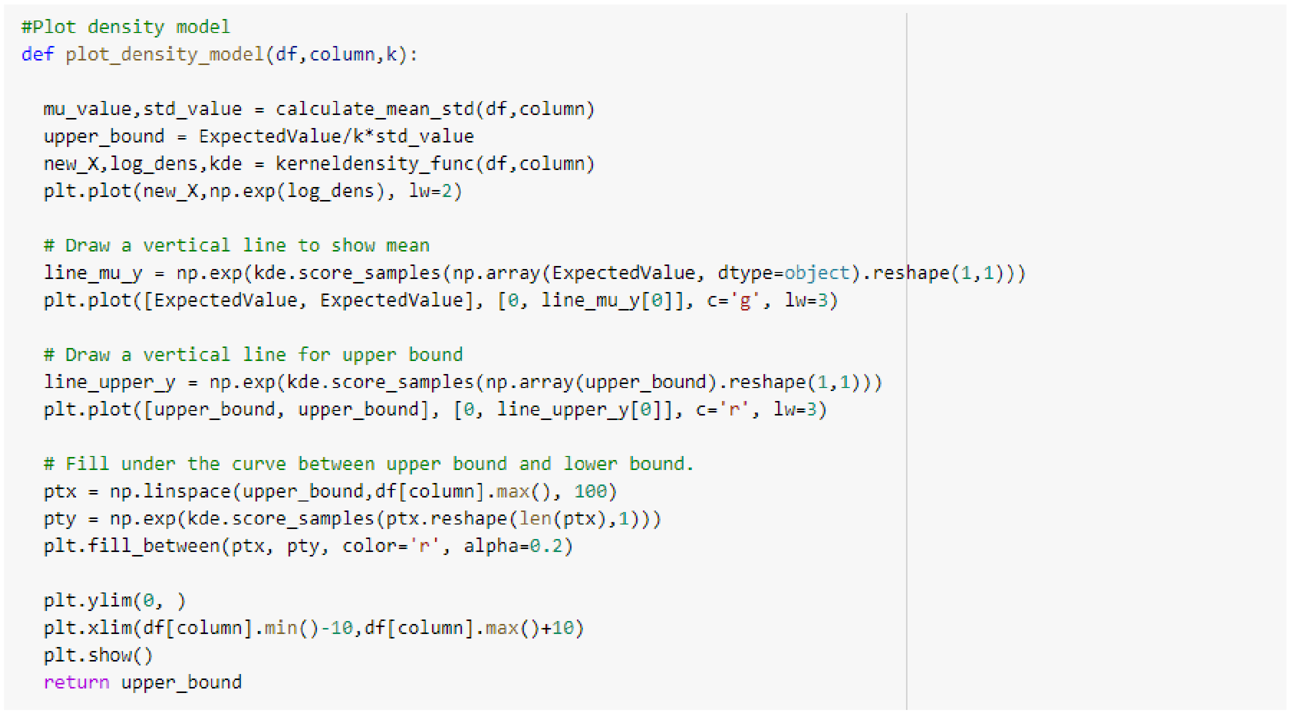
```
#Markov's bounds
def MarkovInequality(df,column,a):


  P_Xsupa = ExpectedValue/a

  print(f"Markov's Inequality \n \t\t\tP(X >= {a}) <= {P_Xsupa}")
  print(f"True value of P is \n \t\t\tP(X) = {true_Proba(a,tb)}")
```

__Function 1: Markov's Inequality__



```python
#kernel density function
def kerneldensity_func(df,column):

  X = df[column].dropna()

  # Convert to numpy ndarray
  X = np.array(X).reshape((len(X), 1))

  #·Fit a Kernel Density Model
  kde = KernelDensity(bandwidth=3).fit(X)

  # Estimate new variables using kde model
  new_X = np.linspace(np.min(X), np.max(X), 1000)[:, np.newaxis]

  log_dens = kde.score_samples(new_X)
  return new_X,log_dens,kde
```

Function 2: Kernel density



```python
#Plot density model
def plot_density_model(df,column,k):

  mu_value,std_value = calculate_mean_std(df,column)
  upper_bound = ExpectedValue/k*std_value
  new_X,log_dens,kde = kerneldensity_func(df,column)
  plt.plot(new_X,np.exp(log_dens), lw=2)

  # Draw a vertical line to show mean
  line_mu_y = np.exp(kde.score_samples(np.array(ExpectedValue, dtype=object).reshape(1,1)))
  plt.plot([ExpectedValue, ExpectedValue], [0, line_mu_y[0]], c='g', lw=3)

  # Draw a vertical line for upper bound
  line_upper_y = np.exp(kde.score_samples(np.array(upper_bound).reshape(1,1)))
  plt.plot([upper_bound, upper_bound], [0, line_upper_y[0]], c='r', lw=3)

  # Fill under the curve between upper bound and lower bound.
  ptx = np.linspace(upper_bound,df[column].max(), 100)
  pty = np.exp(kde.score_samples(ptx.reshape(len(ptx),1)))
  plt.fill_between(ptx, pty, color='r', alpha=0.2)

  plt.ylim(0, )
  plt.xlim(df[column].min()-10,df[column].max()+10)
  plt.show()
  return upper_bound
```

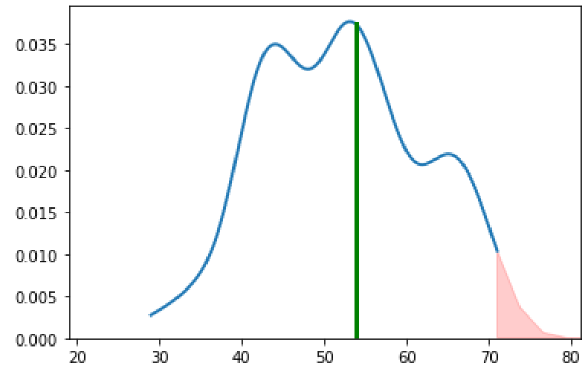
Function 3: Plot density model

Kernel function is placed at each data point and the resulting curves are summed to estimate the overall probability density function as we have a sample with small number of data. As well as smoothing and visualization of our dataset

a. *Application*

```
upper_bound=plot_density_model(df,'age',1.5)
upper_bound
```

```
Mean of the variable age is 52.12162162162162
Standard Deviation of the variable age is 9.765683140330447
```



```
350.71823384640084
```

In the application we will be visualizing the vertical line for the upper bound and fill in the curve of occurrence of an event (a = k * standart deviation).

In this exemple k = 1.5 and the green line is our calculated E[X] expected value.

2. *True probability, and expected value*

```
P = []
N = len(sample1)
#fit value_count to dataframe
values = sample1['age'].value_counts().rename_axis('unique_values').reset_index(name='counts')
tb = pd.DataFrame(values)
for i in range(len(tb)):
  P.append(tb.iloc[i,1]/N)
tb.insert(2,'P',P)
tb
```

	unique_values	counts	P
0	54	8	0.08
1	58	7	0.07
2	41	6	0.06
3	44	5	0.05
4	52	5	0.05
5	59	4	0.04
6	65	4	0.04

Function 4: Probability of each unique value

Calculating the frequency of each unique value in the dataset then dividing it on N and inserting it in the column P.

```
def true_Proba(a,tb):
  true_P = 0.0
  for i in range(len(tb)):
    if(a<tb.iloc[i,0]):
      true_P += tb.iloc[i,2]
  return true_P
```

Function 5: True probability

Sum the total p events by the number of possible outcomes that are superior to an event a.

```
ExpectedValue = 0.0
for i in range(len(tb)):
  ExpectedValue += tb.iloc[i,0]*tb.iloc[i,2]
print(f"E[X] = {ExpectedValue}")
```

```
E[X] = 53.87000000000001
```

Function 6: Expected value

Multiply each value of the random variable by its probability and add the products.

$$E(X) = \mu = \sum x P(x)$$

```
#mean and sdv
def calculate_mean_std(df,column):
  mu_value = df[column].mean()
  std_value = df[column].std()
  print(f"Mean of the variable {column} is {mu_value}")
  print(f"Standard Deviation of the variable {column} is {std_value}")
  return mu_value,std_value
```

Function 7: Mean and std

3. *Tchebyshev's Inequality*

$$\begin{aligned}
\Pr\left(\|X-\mu\|_\alpha \ge k\sigma_\alpha\right) &= \int_\Omega \mathbf{1}_{\|X-\mu\|_\alpha \ge k\sigma_\alpha}\, \mathrm{d}\Pr \\
&= \int_\Omega \left(\frac{\|X-\mu\|_\alpha^2}{\|X-\mu\|_\alpha^2}\right)\cdot \mathbf{1}_{\|X-\mu\|_\alpha \ge k\sigma_\alpha}\, \mathrm{d}\Pr \\
&\le \int_\Omega \left(\frac{\|X-\mu\|_\alpha^2}{(k\sigma_\alpha)^2}\right)\cdot \mathbf{1}_{\|X-\mu\|_\alpha \ge k\sigma_\alpha}\, \mathrm{d}\Pr \\
&\le \frac{1}{k^2\sigma_\alpha^2}\int_\Omega \|X-\mu\|_\alpha^2\, \mathrm{d}\Pr \qquad\qquad \mathbf{1}_{\|X-\mu\|_\alpha \ge k\sigma_\alpha} \le 1 \\
&= \frac{1}{k^2\sigma_\alpha^2}\left(\mathrm{E}\,\|X-\mu\|_\alpha^2\right) \\
&= \frac{1}{k^2\sigma_\alpha^2}\left(\sigma_\alpha^2\right) \\
&= \frac{1}{k^2}
\end{aligned}$$

Figure 2: Bienaymé-Tchebychev's s Inequality

The Chebyshev's inequality is a generalization of the Markov inequality, which is obtained by setting k = 1. Also the Chebyshev's inequality holds for any distribution it is a general inequality that applies to any random variable, regardless of its distribution.

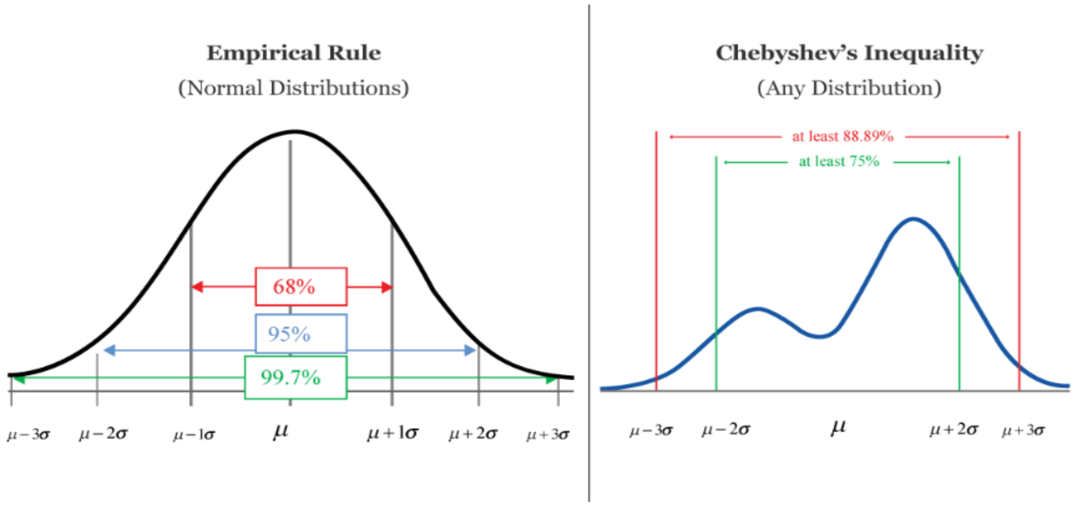


Figure 3: Bienaymé-Tchebychev's s on a districution

```python
def ChebyshevInequality(df,column,a):

  mu_value,std_value = calculate_mean_std(df,column)

  P_Xsupa = round(std_value**2/a**2,2)

  print(f"Chebychev's Inequality \n \t\t\tP(|X - {ExpectedValue}| >= {a}) <= {P_Xsupa}")
  print(f"True value of P is \n \t\t\tP(X) = {true_Proba(a,tb)}")
```

Function 8: Chebyshev's Inequality

b. Application

```python
#chebychev's bounds
def chebychev_bounds(df,column,k):
  mu_value,std_value = calculate_mean_std(df,column)

  upper_bound = mu_value + k * std_value
  lower_bound = mu_value - k * std_value

  print(f"Upper bound for k={k} from the mean is {upper_bound}")
  print(f"Lower bound for k={k} from the mean is {lower_bound}")
  return mu_value,std_value,upper_bound,lower_bound
```

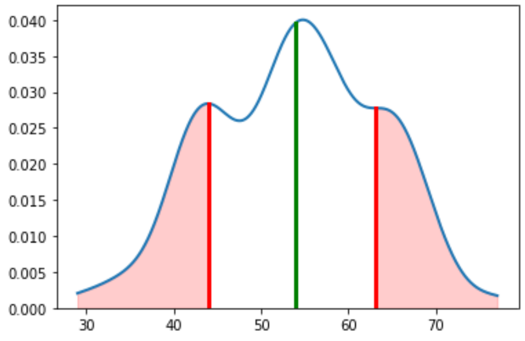
Function 8: Chebyshev's upper and lower bound

```python
#Plot density model
def plot_density_model(df,column,k):
  mu_value,std_value,upper_bound,lower_bound = chebychev_bounds(df,column,k)
  new_X,log_dens,kde = kerneldensity_func(df,column)
  plt.plot(new_X,np.exp(log_dens), lw=2)

  upper_bound = int(upper_bound)
  lower_bound = int(lower_bound)
  # Draw a vertical line to show mean
  line_mu_y = np.exp(kde.score_samples(np.array(mu_value, dtype=object).reshape(1,1)))
  plt.plot([mu_value, mu_value], [0, line_mu_y[0]], c='g', lw=3)

  # Draw a vertical line for upper bound
  line_upper_y = np.exp(kde.score_samples(np.array(upper_bound).reshape(1,1)))
  plt.plot([upper_bound, upper_bound], [0, line_upper_y[0]], c='r', lw=3)

  # Draw a vertical line for lower bound
  line_lower_y = np.exp(kde.score_samples(np.array(lower_bound).reshape(1,1)))
  plt.plot([lower_bound, lower_bound], [0, line_lower_y[0]], c='r', lw=3)

  # Fill under the curve between upper bound and lower bound.
  ptx = np.linspace(lower_bound, upper_bound, 100)
  pty = np.exp(kde.score_samples(ptx.reshape(len(ptx),1)))
  plt.fill_between(ptx, pty, color='r', alpha=0.2)

  plt.ylim(0, )
  plt.show()
  return upper_bound,lower_bound
```

Function 9: plot density model

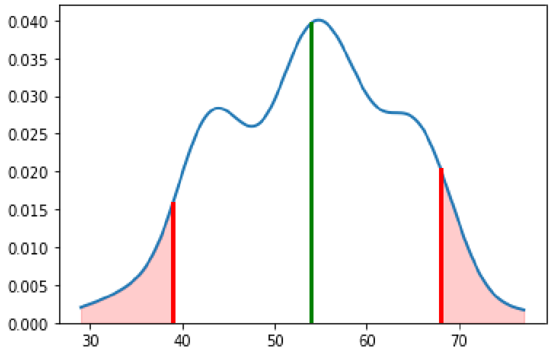
```
[ ]  upper_bound,lower_bound = plot_density_model(sample1,'age',1.0)
```

```
Mean of the variable age is 53.87
Standard Deviation of the variable age is 9.627039991082357
Upper bound for k=1.0 from the mean is 63.49703999108235
Lower bound for k=1.0 from the mean is 44.24296000891764
```



```
[ ]  plot_density_model(df,'age',1.5)
```

```
Mean of the variable age is 53.87
Standard Deviation of the variable age is 9.627039991082357
Upper bound for k=1.5 from the mean is 68.31055998662353
Lower bound for k=1.5 from the mean is 39.429440013376464
```

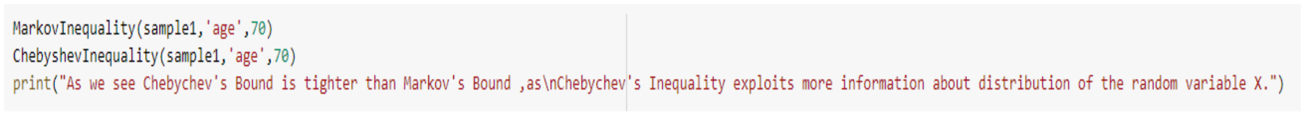


```
(68, 39)
```

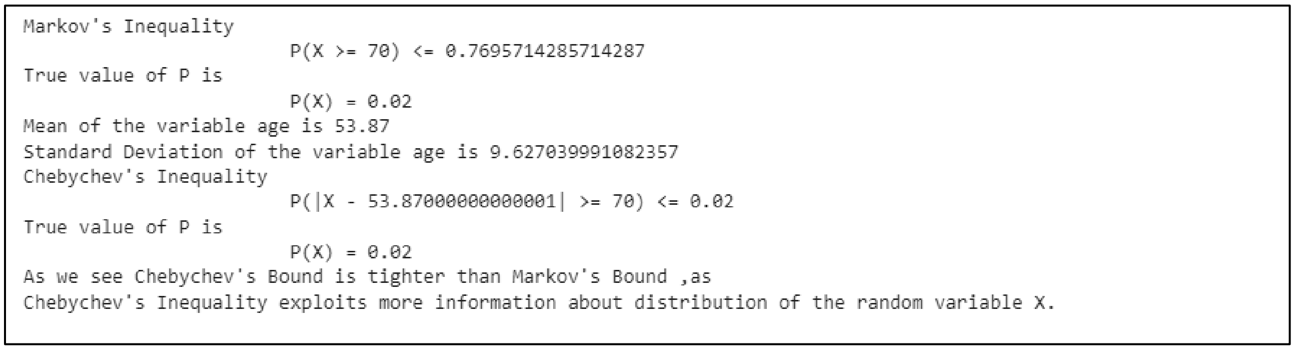
In the application we will be visualizing the vertical line for the upper bound and lower bound and fill in the curve of occurrence of an event (a = k * standart deviation).

In this exemple k = 1.5 and k=1.0 and the green line is our calculated E[X] expected value. As we see the further we are from the expected value the smaller the density gets

4. *Comparison between Markov's and Tchebyshev's inequality*



```
MarkovInequality(sample1,'age',70)
ChebyshevInequality(sample1,'age',70)
print("As we see Chebychev's Bound is tighter than Markov's Bound ,as\nChebychev's Inequality exploits more information about distribution of the random variable X.")
```



```
Markov's Inequality
                        P(X >= 70) <= 0.7695714285714287
True value of P is
                        P(X) = 0.02
Mean of the variable age is 53.87
Standard Deviation of the variable age is 9.627039991082357
Chebychev's Inequality
                        P(|X - 53.87000000000001| >= 70) <= 0.02
True value of P is
                        P(X) = 0.02
As we see Chebychev's Bound is tighter than Markov's Bound ,as
Chebychev's Inequality exploits more information about distribution of the random variable X.
```

In the application The conclusion is by applying both inequalities we got a stronger and a more accurate bound with Tchebyshev's bound, Markov's bound was too far away from the result of true P which is inaccurate for our calculations.

We should note that the key difference between the two inequalities is that Markov's inequality gives a stronger bound when the expected value of the random variable is large, whereas Tchebyshev's inequality gives a stronger bound when the standard deviation of the random variable is small.

III. Conditional expectation (Espérance conditionnelle)

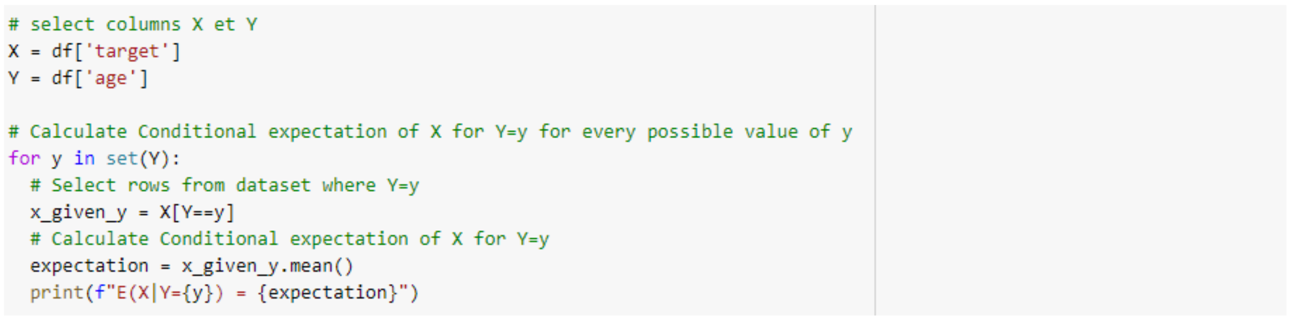
In probability theory, the conditional expectation of a random variable is the expected value of the random variable, given some specific condition.

Formally, the conditional expectation of a random variable X, given a sigma-algebra F, is a function denoted as E[X | F] and is defined as:

$$\begin{aligned}\mathrm{E}(X \mid Y = y) &= \int_{-\infty}^{\infty} x f_{X|Y}(x \mid y)\,\mathrm{d}x \\ &= \frac{1}{f_Y(y)} \int_{-\infty}^{\infty} x f_{X,Y}(x, y)\,\mathrm{d}x.\end{aligned}$$

Figure 4: Conditional expectation

where the sum is over all possible values that the random variable X can take on.



```
# select columns X et Y
X = df['target']
Y = df['age']

# Calculate Conditional expectation of X for Y=y for every possible value of y
for y in set(Y):
  # Select rows from dataset where Y=y
  x_given_y = X[Y==y]
  # Calculate Conditional expectation of X for Y=y
  expectation = x_given_y.mean()
  print(f"E(X|Y={y}) = {expectation}")
```

Function 10: Conditional expectation of heart disease knowing the age

```
E(X|Y=29) = 1.0
E(X|Y=34) = 1.0
E(X|Y=37) = 1.0
E(X|Y=39) = 1.0
E(X|Y=41) = 1.0
E(X|Y=42) = 0.75
E(X|Y=43) = 1.0
E(X|Y=44) = 0.8
E(X|Y=45) = 1.0
E(X|Y=46) = 1.0
E(X|Y=47) = 1.0
E(X|Y=48) = 1.0
E(X|Y=50) = 1.0
E(X|Y=51) = 1.0
E(X|Y=52) = 0.6
E(X|Y=53) = 1.0
E(X|Y=54) = 0.75
E(X|Y=55) = 0.6666666666666666
E(X|Y=56) = 0.25
E(X|Y=57) = 0.5
E(X|Y=58) = 0.7142857142857143
E(X|Y=59) = 0.5
E(X|Y=60) = 0.0
E(X|Y=62) = 0.3333333333333333
E(X|Y=63) = 0.6666666666666666
E(X|Y=64) = 0.6666666666666666
E(X|Y=65) = 0.5
E(X|Y=66) = 0.75
E(X|Y=67) = 0.3333333333333333
E(X|Y=68) = 1.0
E(X|Y=69) = 1.0
E(X|Y=70) = 1.0
E(X|Y=71) = 1.0
E(X|Y=77) = 0.0
```

```python
# select columns X et Y
X = df['age']
Y = df['target']

# Calculate Conditional expectation of X for Y=y for every possible value of y
for y in set(Y):
  # Select rows from dataset where Y=y
  x_given_y = X[Y==y]
  # Calculate Conditional expectation of X for Y=y
  expectation = x_given_y.mean()
  print(f"E(X|Y={y}) = {expectation}")
```

Function 11: Conditional expectation of age knowing the presence or absence of heart disease

```
E(X|Y=0) = 56.569138276553105
E(X|Y=1) = 52.40874524714829
```

IV. Parzen windows approach avec 2D ET 3D:

The Parzen-window method (also known as Parzen-Rosenblatt window method) is a widely used non-parametric approach to estimate a probability density function p(x) for a specific point p(x) from a sample p(xn) that doesn't require any knowledge or assumption about the underlying distribution.

Let R be an elementary region of the feature vector space. The probability that a vector x belongs to the region R is given by:

$$P = \int_{\mathcal{R}} p(\mathbf{x}')d\mathbf{x}'$$

4.1

Suppose there are M independent samples {x1, x2, ..., xM}. The probability that k samples among the M are in the region R follows a binomial law such that:

$$P_k = \binom{M}{k} P^k (1-P)^{M-k}$$

4.2

With :

$$E[k] = MP$$

If the number of observations in R is k, we could estimate P by:

$$P \simeq \frac{k}{M}$$

4.3

Suppose, also, that p(x) is a continuous function on R, we'll have

$$\int_{\mathcal{R}} p(\mathbf{x}')d\mathbf{x}' \simeq p(\mathbf{x}).V$$

4.4

where V is the volume of region R.

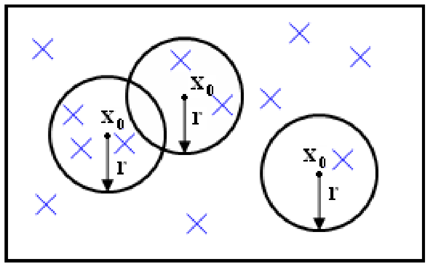
The combination of equations (4.1), (4.3) and (4.4) makes it possible to derive, in the general case,

the estimation of the following density function:

$$\hat{p}_M(\mathbf{x}) = \frac{k_M/M}{V_M}$$

In order to estimate ^pM(x), we use the following approach:

Parzen Window in 2D



same r for any x_0

it consists of choosing a volume VM and calculating the number of samples

that fall in the region R among the k samples

let R be a hypercube in the d-dimensional space.

If hM denotes the length of the side of the hypercube, the volume is given by:

$$V_M = h_M^d$$

To count how many samples fall within this region, and how many lie outside. we would use the following equation to count the samples k_n within this hypercube, where φ is *window function:*

$$\varphi(\frac{\mathbf{x}-\mathbf{x}_i}{h_M}) = 1,$$

if xi belongs to the hypercube centered at x, and zero otherwise.

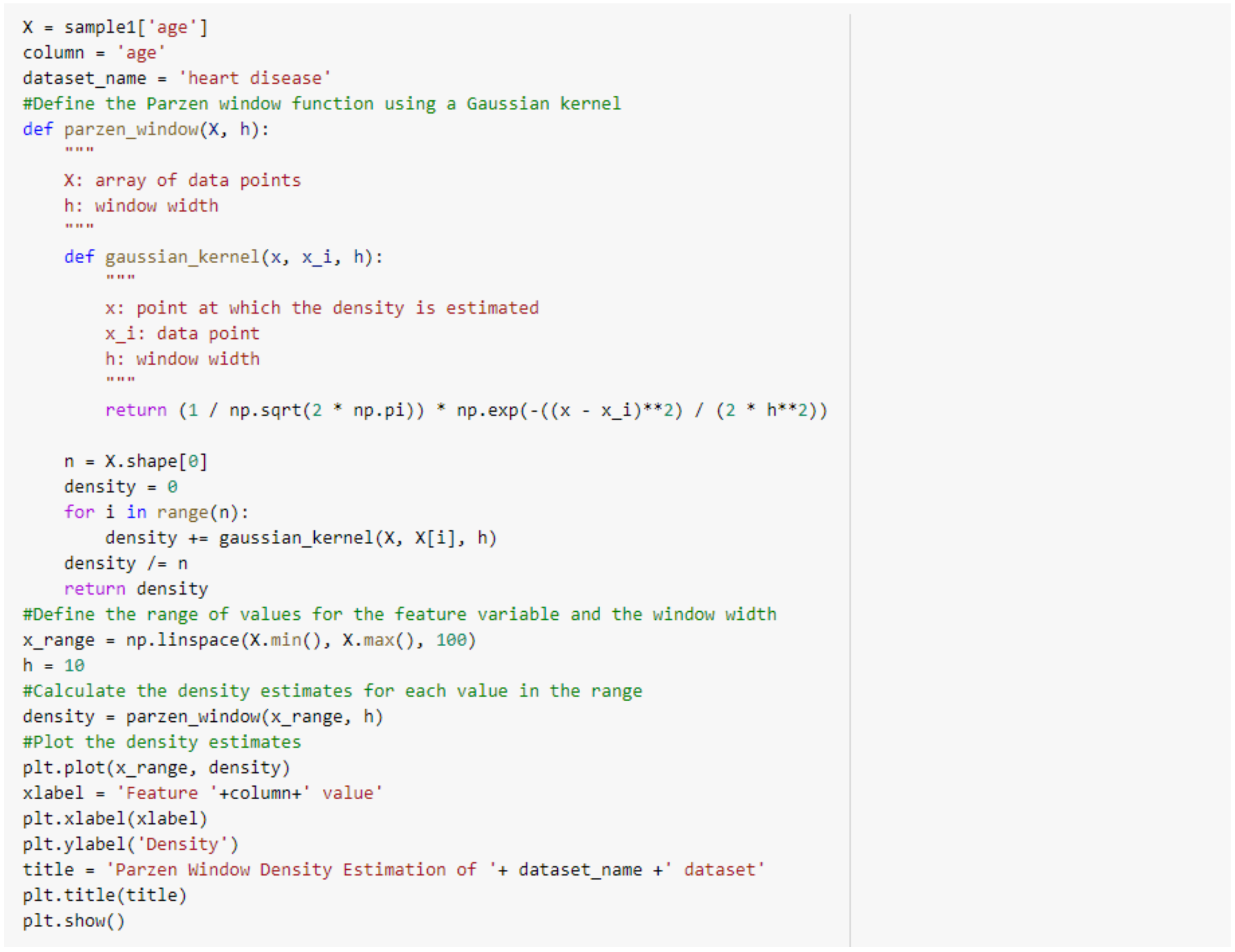
$$\varphi(u) = \begin{cases} 1 & \text{Si } |u_j| \leq 1/2; \qquad j = 1, ..., d \\ 0 & \text{Sinon.} \end{cases}$$

$$k_M = \sum_{i=1}^{M} \varphi(\frac{\mathbf{x}-\mathbf{x}_i}{h_M}).$$

If we extend on this concept, we can define a more general equation, and we obtain the following estimate of pM(x):

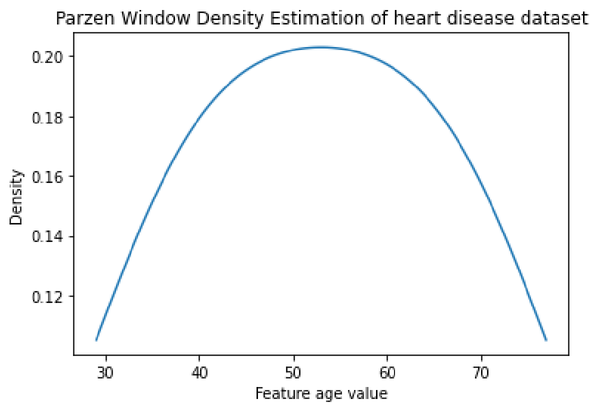
$$\hat{p}_M(\mathbf{x}) = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{h_M^d}\varphi(\frac{\mathbf{x}-\mathbf{x}_i}{h_M}).$$

In term of code, the most common non-parametric approach for estimating the probability density function of a continuous random variable is called kernel smoothing, or kernel density estimation, KDE for short.



```
X = sample1['age']
column = 'age'
dataset_name = 'heart disease'
#Define the Parzen window function using a Gaussian kernel
def parzen_window(X, h):
    """
    X: array of data points
    h: window width
    """
    def gaussian_kernel(x, x_i, h):
        """
        x: point at which the density is estimated
        x_i: data point
        h: window width
        """
        return (1 / np.sqrt(2 * np.pi)) * np.exp(-((x - x_i)**2) / (2 * h**2))

    n = X.shape[0]
    density = 0
    for i in range(n):
        density += gaussian_kernel(X, X[i], h)
    density /= n
    return density
#Define the range of values for the feature variable and the window width
x_range = np.linspace(X.min(), X.max(), 100)
h = 10
#Calculate the density estimates for each value in the range
density = parzen_window(x_range, h)
#Plot the density estimates
plt.plot(x_range, density)
xlabel = 'Feature '+column+' value'
plt.xlabel(xlabel)
plt.ylabel('Density')
title = 'Parzen Window Density Estimation of '+ dataset_name +' dataset'
plt.title(title)
plt.show()
```

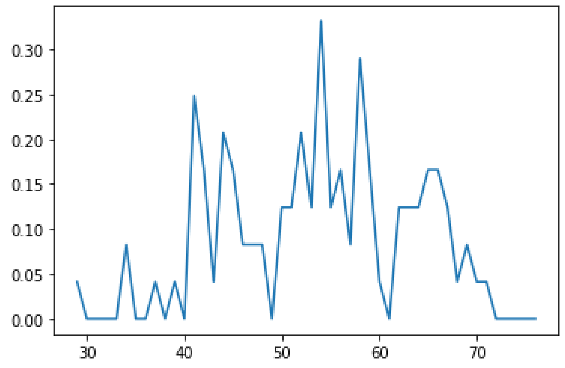
Function 12: Parzen window with a gaussian kernel and a band width value of 10



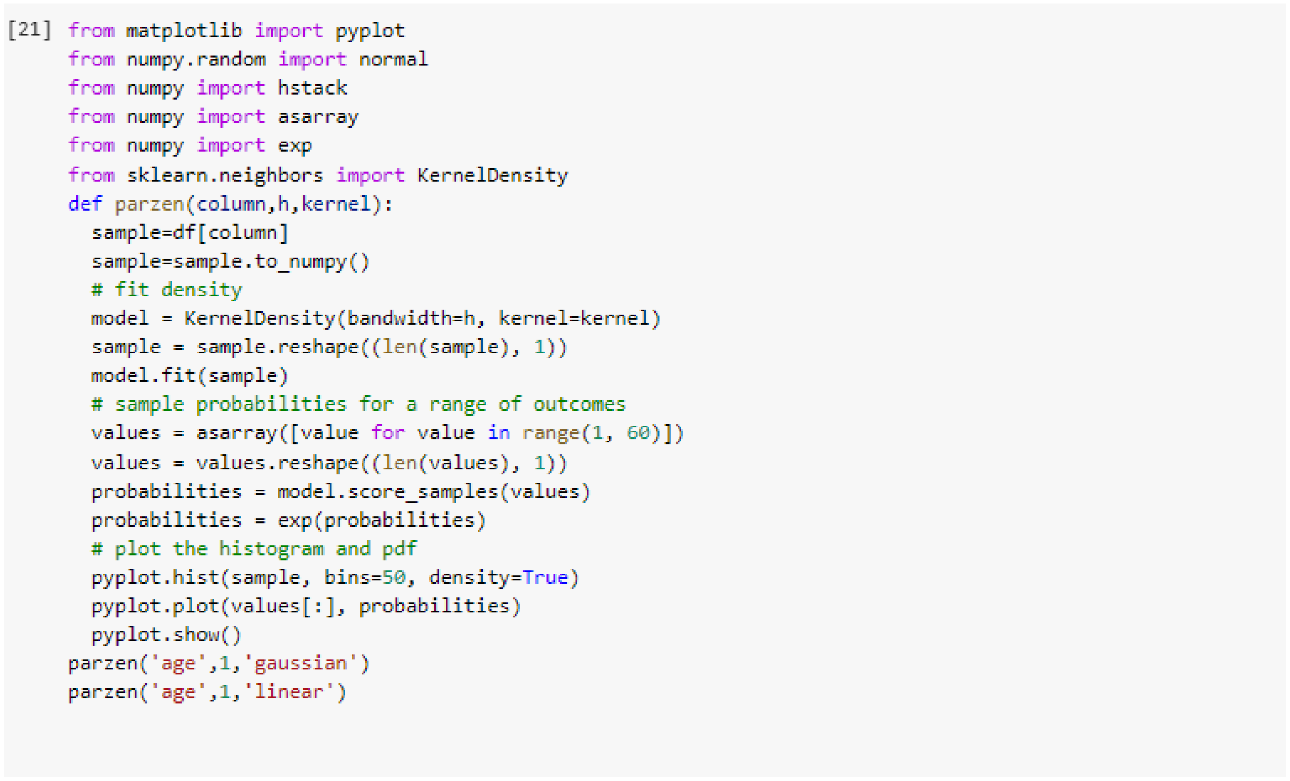
A parameter, called the *smoothing parameter* or the *bandwidth* (h), controls the scope, or window of observations, from the data sample that contributes to estimating the probability for a given sample.

```
from scipy.stats import gaussian_kde

# Extract the age feature
ages = sample1['age'].values

# Estimate the PDF of the age feature using Parzen window estimation
kde = gaussian_kde(ages,bw_method=0.01)

# Evaluate the PDF at a range of ages
age_range = range(int(min(ages)), int(max(ages)))
pdf = kde.evaluate(age_range)

# Plot the PDF
import matplotlib.pyplot as plt
plt.plot(age_range, pdf)
plt.show()
```

Function 13: Parzen window with a gaussian kernel and a band width value of 0.1

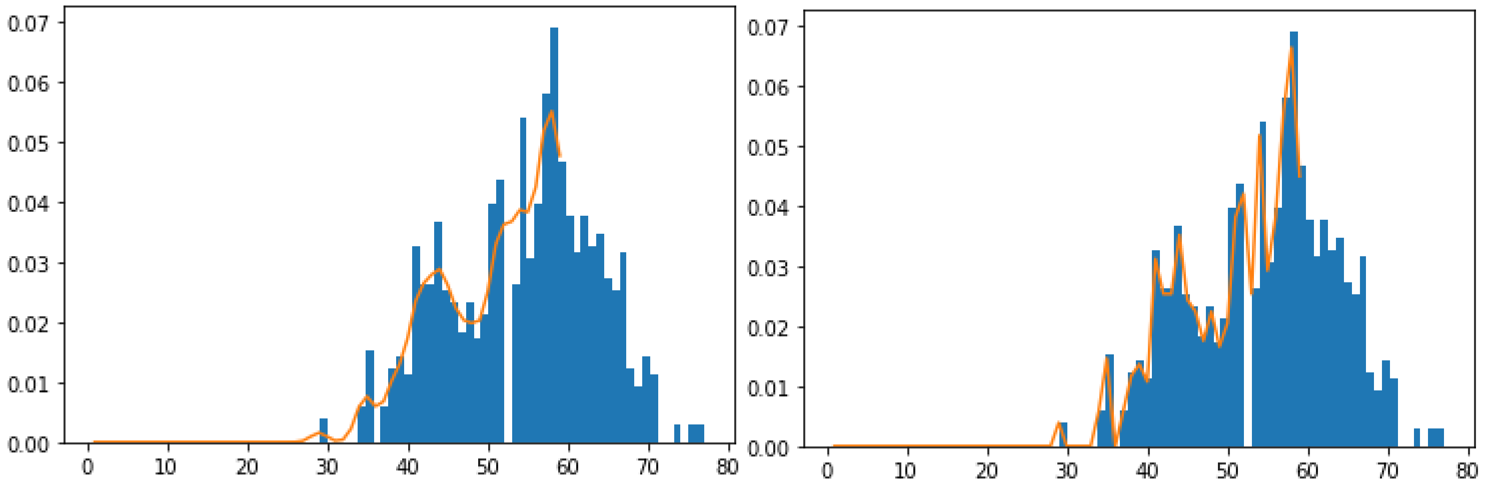


A large window may result in a coarse density with little details, whereas a small window may have too much detail and not be smooth or general enough to correctly cover new or unseen examples.



```
[21] from matplotlib import pyplot
     from numpy.random import normal
     from numpy import hstack
     from numpy import asarray
     from numpy import exp
     from sklearn.neighbors import KernelDensity
     def parzen(column,h,kernel):
       sample=df[column]
       sample=sample.to_numpy()
       # fit density
       model = KernelDensity(bandwidth=h, kernel=kernel)
       sample = sample.reshape((len(sample), 1))
       model.fit(sample)
       # sample probabilities for a range of outcomes
       values = asarray([value for value in range(1, 60)])
       values = values.reshape((len(values), 1))
       probabilities = model.score_samples(values)
       probabilities = exp(probabilities)
       # plot the histogram and pdf
       pyplot.hist(sample, bins=50, density=True)
       pyplot.plot(values[:], probabilities)
       pyplot.show()
     parzen('age',1,'gaussian')
     parzen('age',1,'linear')
```

Function 14: Parzen window with two different kernels with the same band width

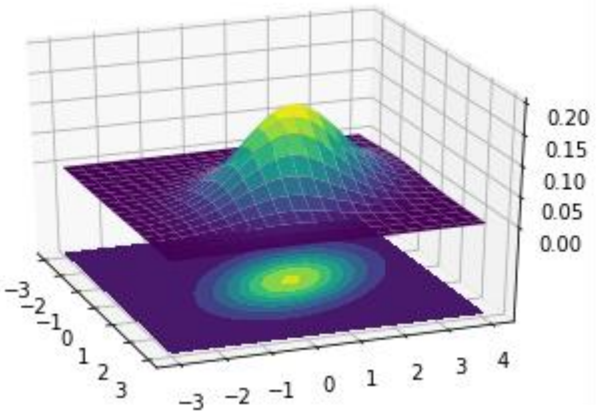


One key difference between the Gaussian kernelon the left and the linear kernelon the right is that the former is able to capture nonlinear relationships in the data, while the latter can only model linear relationships.

The smaller the band width is the more gaussian kernel acts like a linear kernel

```
# Our 2-dimensional distribution will be over variables X and Y
N = 60
X = np.linspace(-3, 3, N)
Y = np.linspace(-3, 4, N)
X, Y = np.meshgrid(X, Y)

# Mean vector and covariance matrix
mu = np.array([0., 1.])
Sigma = np.array([[ 1. , -0.5], [-0.5,  1.5]])

# Pack X and Y into a single 3-dimensional array
pos = np.empty(X.shape + (2,))
pos[:, :, 0] = X
pos[:, :, 1] = Y

# The distribution on the variables X, Y packed into pos.
F = multivariate_normal(mu, Sigma)
Z = F.pdf(pos)

# Create a surface plot and projected filled contour plot under it.
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X, Y, Z, rstride=3, cstride=3, linewidth=1, antialiased=True,
                cmap=cm.viridis)

cset = ax.contourf(X, Y, Z, zdir='z', offset=-0.15, cmap=cm.viridis)

# Adjust the limits, ticks and view angle
ax.set_zlim(-0.15,0.2)
ax.set_zticks(np.linspace(0,0.2,5))
ax.view_init(27, -21)

plt.show()
```

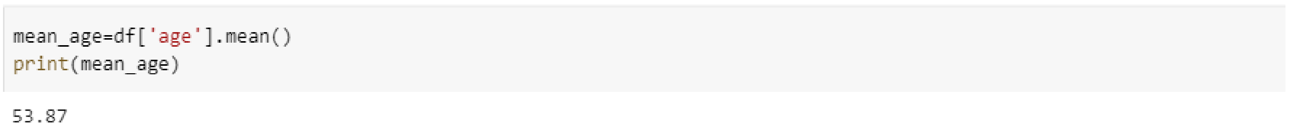
Function 15: 3D Parzen window on a generated dataset



V. Estimations:

a. *Point estimation*

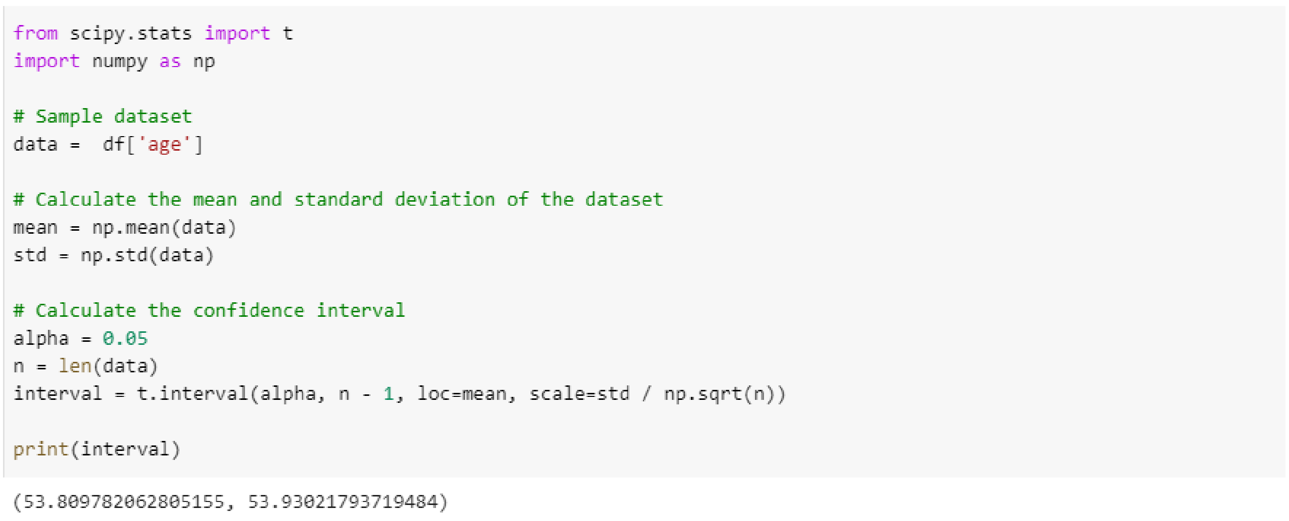
Point estimation, in statistics, the process of finding an approximate value of some parameter—such as the mean (average)—of a population from random samples of the population.



```
mean_age=df['age'].mean()
print(mean_age)
```

```
53.87
```

b. *Estimation of confidence intervals*



```
from scipy.stats import t
import numpy as np

# Sample dataset
data =  df['age']

# Calculate the mean and standard deviation of the dataset
mean = np.mean(data)
std = np.std(data)

# Calculate the confidence interval
alpha = 0.05
n = len(data)
interval = t.interval(alpha, n - 1, loc=mean, scale=std / np.sqrt(n))

print(interval)
```

```
(53.809782062805155, 53.93021793719484)
```

Function 17: confidence interval

c. *Likelihood and Maximum Likelihood Estimation*

The normal density function is given by,

$$\ell = \prod_{i=1}^{N} f(x_i)$$

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

The likelihood of an observed x-value, say x1 is simply □=f(x1). If there are multiple values, x1,x2,x3,□,xN, the likelihood is given by the product of the individual likelihoods,

We want to find μ and σ given a bunch of x-values and assuming a normal distribution. We do this my choosing μ and σ as to maximize the likelihood. In practice, one actually maximizes the log of the likelihood as this doesn't affect the estiamted values of our unknown parameters, but does simplify the calculation, both analytically and numerically.

$$\frac{d\ln(\ell)}{d\mu} = 0 \qquad \frac{d\ln(\ell)}{d\sigma} = 0$$

We find the maximum by setting the derivatives equal to zero:

After solving we get:

$$\mu^* = \frac{1}{N}\sum_{i=1}^{N} x_i$$

and

$$\sigma^* = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu^*)^2}$$

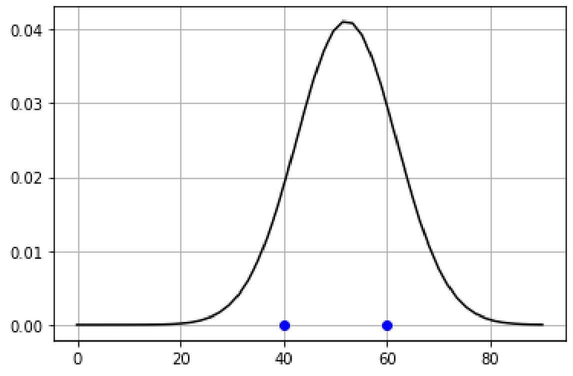
- The results for the normal distribution turn out to be the sample mean and standard deviation

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from scipy.optimize import minimize
```

```
mu = np.mean(df['age'])
sigma = np.std(df['age'])


x = np.linspace(0, 90)
y = norm.pdf(x, loc = mu, scale = sigma)

plt.plot(x, y, 'k')
plt.grid(True)

x1 = 40
x2 = 60

plt.plot(x1, 0, 'bo')
plt.plot(x2, 0, 'bo')
#plt.savefig('normal.png')

l1 = norm.pdf(x1, loc = mu, scale = sigma)
l2 = norm.pdf(x2, loc = mu, scale = sigma)

print(l1)
print(l2)

print(l1 * l2)
```

Function 18: probability of x1 and x2 on a normal distribution

```
0.018837427541982154
0.029573292320968374
0.000557084751274099
```



For example we calculate the likelihood of two points x1 and x2 and we calculate the product

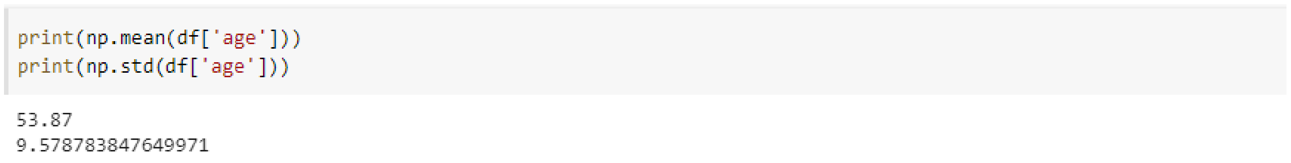
The product (l1*l2) can get smaller with each time we do more multiplication

```
def log_likelihood(p, x):
    mu = df['age'].mean()
    sigma = df['age'].std()

    l = np.sum( np.log( norm.pdf(x, loc = mu, scale = sigma) ) )

    return -l

def constraint(p):
    sigma = p[1]

    return sigma
```

Function 19: logarithmic likelihood

Now we need to take the log of the density and then do the sum

There is no maximum function in scipy there is a minimum function and to find the maximum we will need to find the minimum and return the negative value of it



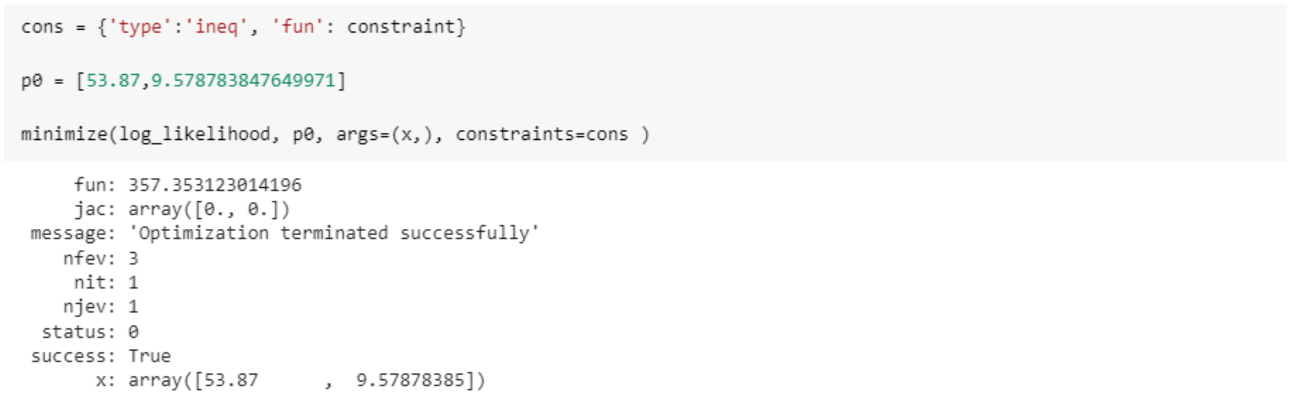
```
print(np.mean(df['age']))
print(np.std(df['age']))
```

```
53.87
9.578783847649971
```

We already have our mean and standard deviation values from the heart disease dataset to begin with and compare at the end

d. Application:

First we need to create a dictionary of constraints to tell the solver that we are doing an inequality function



```
cons = {'type':'ineq', 'fun': constraint}

p0 = [53.87,9.578783847649971]

minimize(log_likelihood, p0, args=(x,), constraints=cons )
```

```
     fun: 357.353123014196
     jac: array([0., 0.])
 message: 'Optimization terminated successfully'
    nfev: 3
     nit: 1
    njev: 1
  status: 0
 success: True
       x: array([53.87      ,  9.57878385])
```

Function 20: optimization of likelihood value

After executing we find the same exact values as the mean and the standard deviation