1.

a)
The Probability density refers to the probability that a continuous random variable x will exist within a set of conditions.

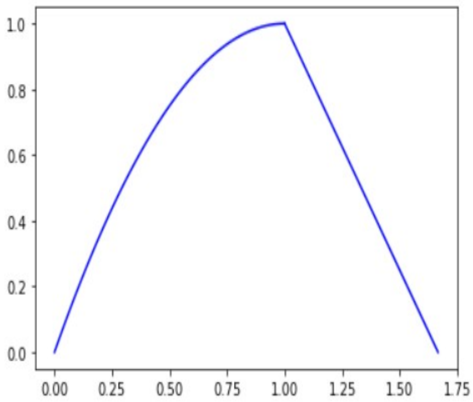
b)

```
[140]: import matplotlib.pyplot as plt
       import numpy as np


       x1 = np.linspace(0, 1, 100)
       x2 = np.linspace(1, 5/3, 100)


       y = 2 * x1 - x1**2
       a = 5/2 - 3/2 * x2


       plt.plot(x1, y, "blue")
       plt.plot(x2, a, "blue")
       plt.show()
```



c)
The probability density function (PDF) f(x) of a continuous distribution is defined as the derivative of the (cumulative) distribution function.

The probability density function of a continuous random variable x is a function

$f_X : \mathbb{R} \to [0, \infty)$ such that $P(X \in [a,b]) = \int_a^b f_X(x)dx$ for any interval $[a,b] \subseteq \mathbb{R}$.

The function must meet the following conditions:

1. f(x) >=0 for all values of x
2. The $\int_{-\infty}^{\infty} f(x)dx=1$

d)
To be sure, that the Probability density refers to the probaboloty that the random variable X is continuous and does exist. It automatically follows, to use intervals to get the continuity.

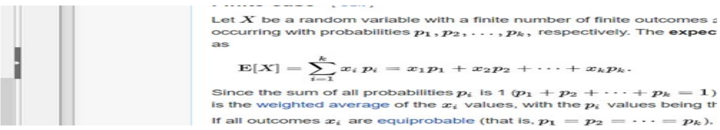
e)
The cumulative distribution function (cdf) is the probability that the variable takes a value less than or equal to x.
That is **F(x)=Pr[X≤x]=α**

For a continuous distribution, this can be expressed mathematically as **F(x)=∫x−∞f(μ)dμ**

For a discrete distribution, the cdf can be expressed as **F(x)=∑xi=0f(i)**

f)

g) When X is a random variable with a finite number of also finite outcomes (x1, x2, ...xk) occuring with probabilitys (p1, p2, ...pk). The expectation of X is defined as:

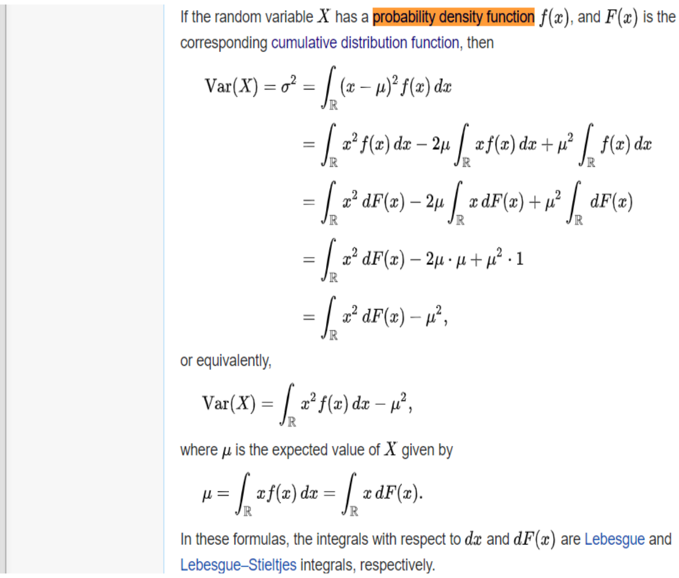


Let X be a random variable with a finite number of finite outcomes
occurring with probabilities $p_1, p_2, \ldots, p_k$, respectively. The **expec**
as

$$\mathrm{E}[X] = \sum_{i=1}^{k} x_i\, p_i = x_1 p_1 + x_2 p_2 + \cdots + x_k p_k.$$

Since the sum of all probabilities p_i is 1 ($p_1 + p_2 + \cdots + p_k = 1$)
is the weighted average of the x_i values, with the p_i values being th
If all outcomes x_i are equiprobable (that is, $p_1 = p_2 = \cdots = p_k$).

j) VAR(x) is:



If the random variable X has a probability density function $f(x)$, and $F(x)$ is the corresponding cumulative distribution function, then

$$\begin{aligned}
\operatorname{Var}(X) = \sigma^2 &= \int_{\mathbb{R}} (x-\mu)^2 f(x)\, dx \\
&= \int_{\mathbb{R}} x^2 f(x)\, dx - 2\mu \int_{\mathbb{R}} x f(x)\, dx + \mu^2 \int_{\mathbb{R}} f(x)\, dx \\
&= \int_{\mathbb{R}} x^2\, dF(x) - 2\mu \int_{\mathbb{R}} x\, dF(x) + \mu^2 \int_{\mathbb{R}} dF(x) \\
&= \int_{\mathbb{R}} x^2\, dF(x) - 2\mu \cdot \mu + \mu^2 \cdot 1 \\
&= \int_{\mathbb{R}} x^2\, dF(x) - \mu^2,
\end{aligned}$$

or equivalently,

$$\operatorname{Var}(X) = \int_{\mathbb{R}} x^2 f(x)\, dx - \mu^2,$$

where μ is the expected value of X given by

$$\mu = \int_{\mathbb{R}} x f(x)\, dx = \int_{\mathbb{R}} x\, dF(x).$$

In these formulas, the integrals with respect to dx and $dF(x)$ are Lebesgue and Lebesgue–Stieltjes integrals, respectively.

2.

a)

The function should only be derivated on x because of f(x) ?

nx	nx	nx
ny	ny	ny
nz	nz	nz

5.

a)

RANSAC is an algorithm to eliminate bad located points or wrong point correspondences.

p is the lowest number of needed point correspondences to estimate the model:

1. Choose random m Sample Sets with p Point correspondences.
2. Calculate this model per Set i.
3. Define the number of consistent and not consistent correspondences (consistent if Error<Threshold)
4. Memorize the biggest set with the most consistent points.
5. Reestimate the model with all consistent correspondences.

b)

A breakpoint is the value corresponding to the number of points k for which a data set exists, which can’t be shattered by the hypothesis set HH. The hypothesis set is not able to describe all possible splits over the data points and mH(N) ≤ 2km H(N) ≤ 2k.

c)

The breakpoint for the RANSAC estimator is ~(2)p

d)

```
import numpy as np
from matplotlib import pyplot as plt

from sklearn import linear_model, datasets


n_samples = 1000
n_outliers = 50


X, y, coef = datasets.make_regression(n_samples=n_samples, n_features=1,
                                      n_informative=1, noise=10,
                                      coef=True, random_state=0)
```

```
# Add outlier
np.random.seed(0)
X[:n_outliers] = 4 + 0.6 * np.random.normal(size=(n_outliers, 1))
y[:n_outliers] = -4 + 12 * np.random.normal(size=n_outliers)

# Fit line using the data
lr = linear_model.LinearRegression()
lr.fit(X, y)
ransac = linear_model.RANSACRegressor()
ransac.fit(X, y)
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)

# Predict data of estimated models
line_X = np.arange(X.min(), X.max())[:, np.newaxis]
line_y = lr.predict(line_X)
line_y_ransac = ransac.predict(line_X)

# Compare estimated coefficients
print("Estimated coefficients (true, linear regression, RANSAC):")
print(coef, lr.coef_, ransac.estimator_.coef_)

lw = 2
plt.scatter(X[inlier_mask], y[inlier_mask], color='yellowgreen', marker='.',
            label='Inliers')
plt.scatter(X[outlier_mask], y[outlier_mask], color='gold', marker='.',
            label='Outliers')
plt.plot(line_X, line_y, color='navy', linewidth=lw, label='Linear regressor')
plt.plot(line_X, line_y_ransac, color='cornflowerblue', linewidth=lw,
         label='RANSAC regressor')
plt.legend(loc='lower right')
plt.xlabel("Input")
plt.ylabel("Response")
plt.show()
```