Banha University
Faculty of Engineering
Department of Computer Engineering

Course: EMP301 (Engineering Mathematics (5))
Term: 241
Course Project cover page

S#	Student Name	Edu Email	Student ID	Marks		
				Report & Slides (30)	Implementation (50)	Presentation (20)
1	Abdelwahab Alaa	abdulwahab402664@feng.bu.edu.eg	231903630			
2	Rana Ehab	rana406651@feng.bu.edu.eg	221903164			
3	Shahd Amgad	shahd402802@feng.bu.edu.eg	221903116			
4	Omar Mohamed Adel	omar.mohamed21@feng.bu.edu.eg	221902996			
5	Shreif Mostafa Samy	sherif414190@feng.bu.edu.eg	221903216			
6						
7						
8						

Date handed in: 19 / 12 / 2023.

Report & Slides (35)		Points
1. Content	Project includes all material needed to give a good understanding of the topic.	10
	All content throughout the paper & presentation is accurate. There are no factual errors.	10
2. Format	Sequencing of Information: Information is organized in a clear, logical way. It is easy to anticipate the next slide.	5
	All graphics are attractive (size and colors) and support the topic of the presentation.	5
	Text-Font Choice & Formatting: Font formats (color, bold, italic) have been carefully planned to enhance readability and content.	5
Presentation (20)		
1. Delivery	Members understand presented material and can clearly answer all questions about it.	10
	Members spoke at a good rate, volume and with good grammar. They maintained eye-contact while using, but not reading their notes.	5
2. Cooperation	Group's members share tasks and all performed responsibly all of the time.	5
Implementation (45)		
1. Code	Members understand the code and can clearly answer all questions about it.	20
	The code is completely functional and correctly producing the outputs.	25
Total Score		**100**

Table of Contents:

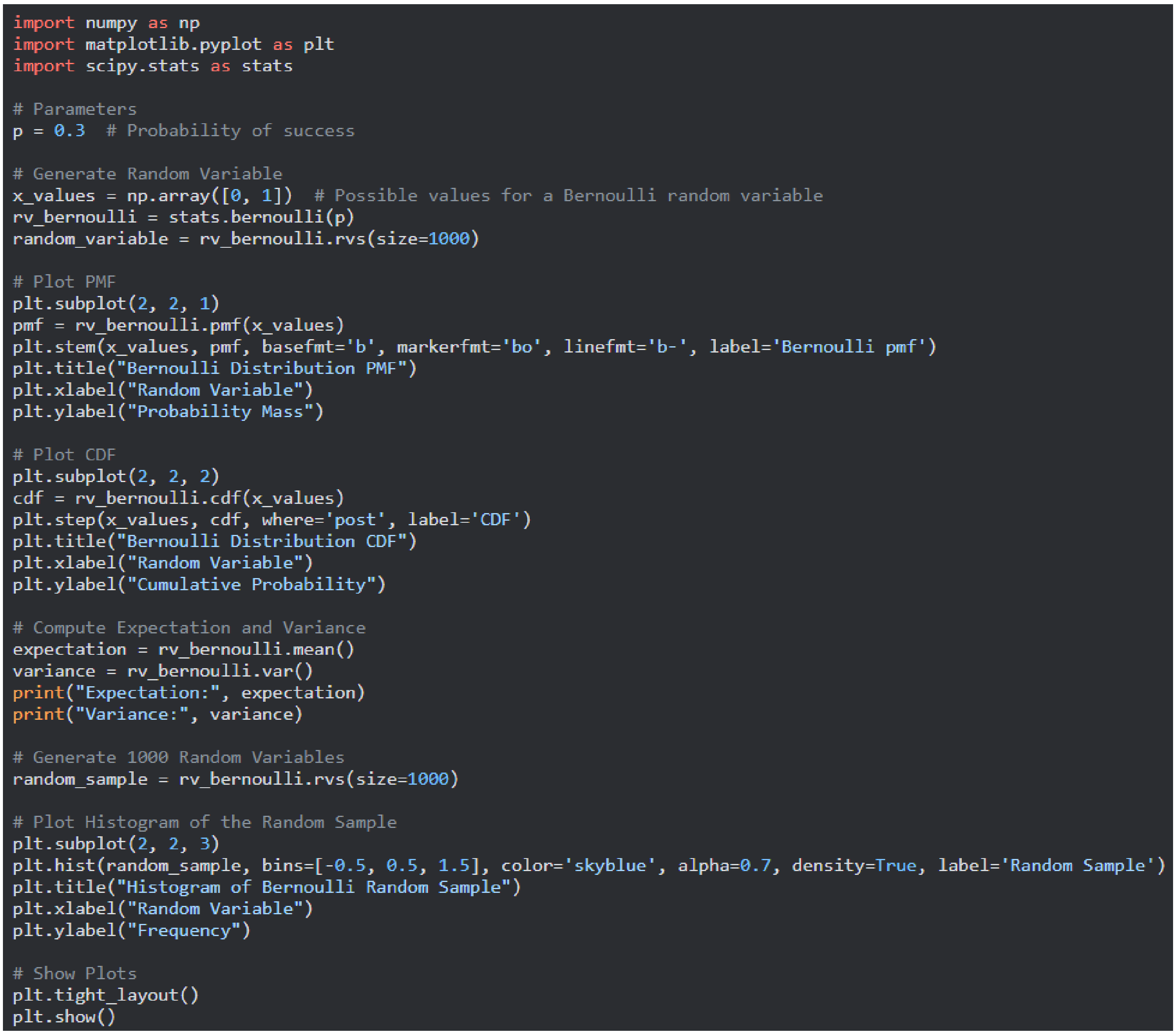
Nomenclature:

	Mean	Variance
Random Variable	μ	σ^2
Bernoulli	P	p(1-p)
Binomial	np	np(1-p)
Geometric	$\frac{1}{p}$	$\frac{1-p}{p^2}$
Exponential	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
Uniform	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
Poisson	λ	λ
Gaussain	μ	σ^2

Discrete random variables

Bernoulli distribution

Bernoulli Distribution is s a special kind of distribution that is used to model real-life examples and can be used in many different types of applications. A random experiment that can only have an outcome of either 1 or 0 is known as a Bernoulli trial. Such an experiment is used in a Bernoulli distribution.



```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

# Parameters
p = 0.3  # Probability of success

# Generate Random Variable
x_values = np.array([0, 1])  # Possible values for a Bernoulli random variable
rv_bernoulli = stats.bernoulli(p)
random_variable = rv_bernoulli.rvs(size=1000)

# Plot PMF
plt.subplot(2, 2, 1)
pmf = rv_bernoulli.pmf(x_values)
plt.stem(x_values, pmf, basefmt='b', markerfmt='bo', linefmt='b-', label='Bernoulli pmf')
plt.title("Bernoulli Distribution PMF")
plt.xlabel("Random Variable")
plt.ylabel("Probability Mass")

# Plot CDF
plt.subplot(2, 2, 2)
cdf = rv_bernoulli.cdf(x_values)
plt.step(x_values, cdf, where='post', label='CDF')
plt.title("Bernoulli Distribution CDF")
plt.xlabel("Random Variable")
plt.ylabel("Cumulative Probability")

# Compute Expectation and Variance
expectation = rv_bernoulli.mean()
variance = rv_bernoulli.var()
print("Expectation:", expectation)
print("Variance:", variance)

# Generate 1000 Random Variables
random_sample = rv_bernoulli.rvs(size=1000)

# Plot Histogram of the Random Sample
plt.subplot(2, 2, 3)
plt.hist(random_sample, bins=[-0.5, 0.5, 1.5], color='skyblue', alpha=0.7, density=True, label='Random Sample')
plt.title("Histogram of Bernoulli Random Sample")
plt.xlabel("Random Variable")
plt.ylabel("Frequency")

# Show Plots
plt.tight_layout()
plt.show()
```

- First we begin with Importing Libraries:

begin by importing the necessary libraries: numpy, matplotlib.pyplot, and scipy stats

- Second we define some Parameters:

The code defines a parameter p which represents the probability of success for a Bernoulli random variable.

- Generating Random Variables:

we create an array x_values to represent the possible values of the Bernoulli random variable (0 and 1). then create a Bernoulli random variable object rv_bernoulli using scipy.stats.bernoulli and generate a random sample of size 1000 from the distribution using the rvs method.

- Plotting PMF:

set up a subplot with plt.subplot(2, 2, 1) to create a 2x2 grid of plots and selects the first subplot. then compute the probability mass function (PMF) of the Bernoulli distribution using the pmf method of the rv_bernoulli object and plots it using plt.stem. The stem plot visualizes the probability mass at each value. The plt.title, plt.xlabel, and plt.ylabel functions are used to set the title and axis labels for the plot.

- Plotting CDF:

Similar to the PMF plot, set up the second subplot and computes the cumulative distribution function (CDF) of the Bernoulli distribution using the cdf method of the rv_bernoulli object. then plot the CDF using plt.step. The where='post' argument specifies that the steps should be placed after the x-values. Then use the plt.title, plt.xlabel, and plt.ylabel to set the title and axis labels for the plot.

- Computing Expectation and Variance:

we compute the expectation (mean) and variance of the Bernoulli distribution using the mean and var methods of the rv_bernoulli object, respectively. The values are stored in the variables expectation and variance and then printed.

- Generating Random Sample:

The code generates another random sample of size 1000 from the Bernoulli distribution using the rvs method and stores it in the variable random_sample.

- Plotting Histogram:

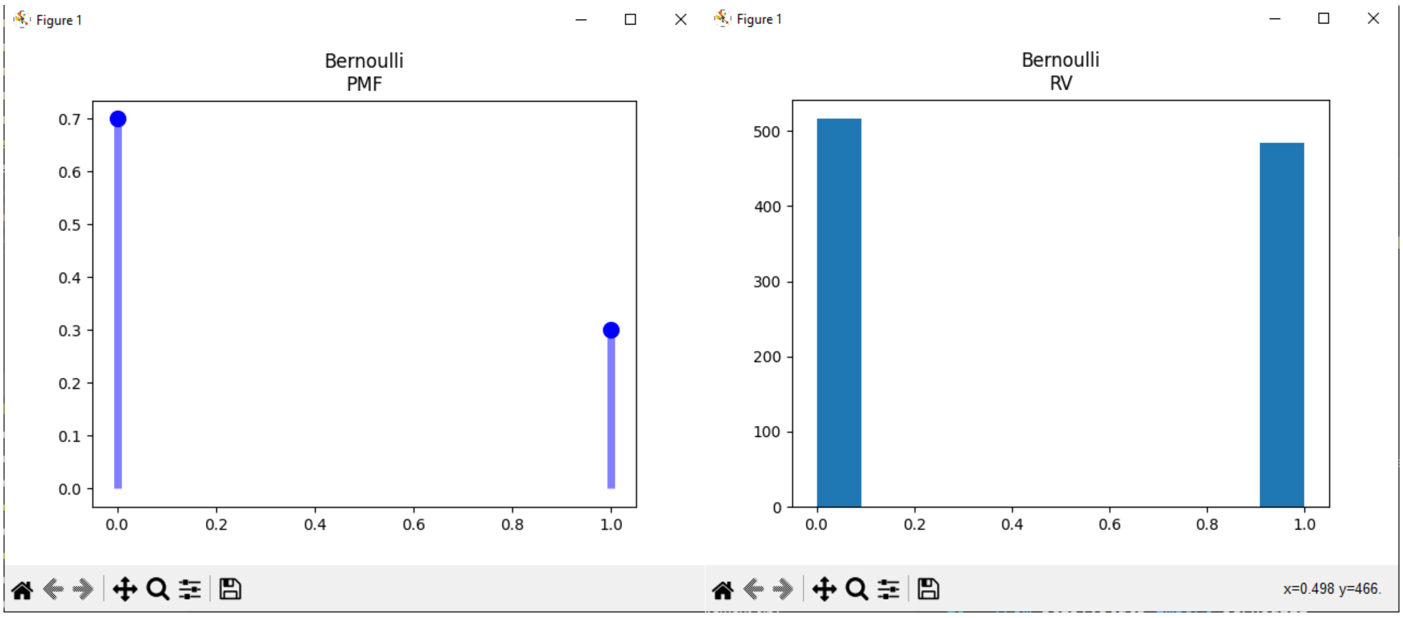
set up the third subplot and plots a histogram of the random sample using plt.hist. The bins argument specifies the bin edges for the histogram, and density=True normalizes the histogram so that the area under the histogram is equal to 1. The plot title and axis labels are set.

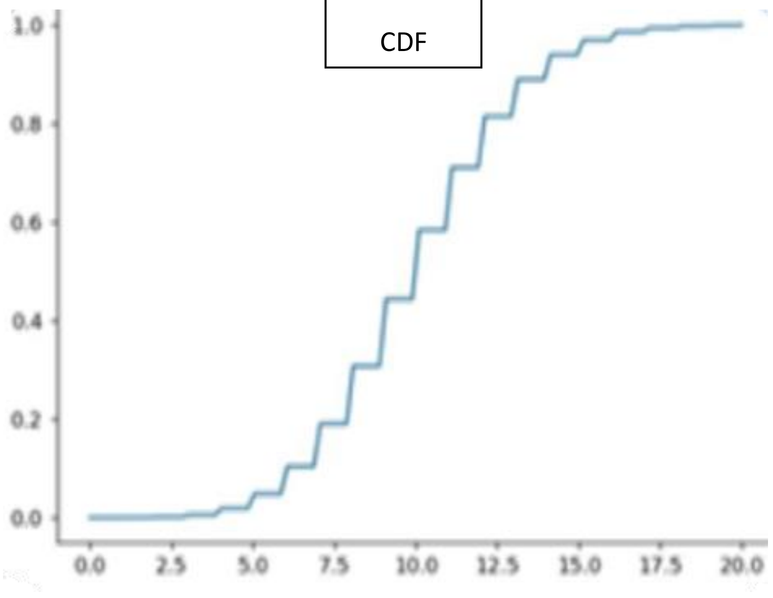
- Displaying Plots:

The code calls plt.tight_layout() to adjust the spacing between subplots and plt.show() to display the plots on the screen.

Overall, this code generates and visualizes a Bernoulli distribution, computes its expectation and variance, generates a random sample, and plots a histogram of the sample.

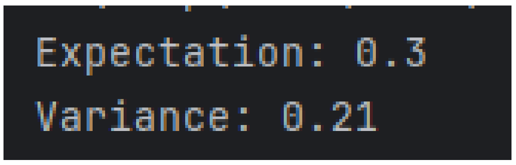
Run of code:





Bernoulli

CDF



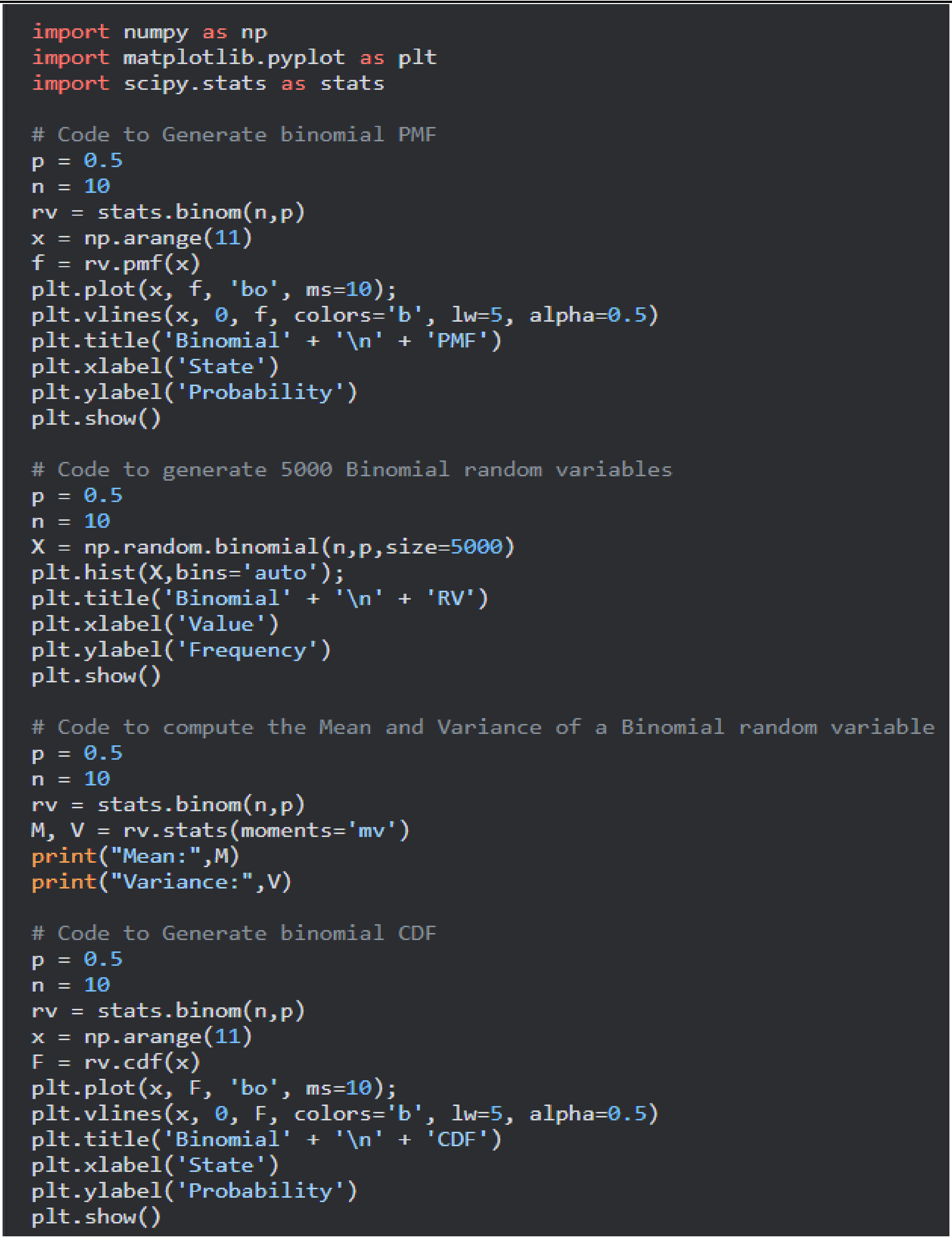
```
Expectation: 0.3
Variance: 0.21
```

Binomial distribution

Binomial distribution is one of the most popular distributions in statistics, along with normal distribution. Binomial distribution is a discrete probability distribution of several successes (X) in a sequence of independent experiments (n). Each experiment has two possible outcomes: success and failure. Success outcome has a probability p, and failure has probability (1−p)

$$P(x) = \binom{n}{x} p^x q^{n-x} = \frac{n!}{(n-x)!\,x!} p^x q^{n-x}$$

where
n = the number of trials (or the number being sampled)
x = the number of successes desired
p = probability of getting a success in one trial
q = 1 - p = the probability of getting a failure in one trial



```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

# Code to Generate binomial PMF
p = 0.5
n = 10
rv = stats.binom(n,p)
x = np.arange(11)
f = rv.pmf(x)
plt.plot(x, f, 'bo', ms=10);
plt.vlines(x, 0, f, colors='b', lw=5, alpha=0.5)
plt.title('Binomial' + '\n' + 'PMF')
plt.xlabel('State')
plt.ylabel('Probability')
plt.show()

# Code to generate 5000 Binomial random variables
p = 0.5
n = 10
X = np.random.binomial(n,p,size=5000)
plt.hist(X,bins='auto');
plt.title('Binomial' + '\n' + 'RV')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

# Code to compute the Mean and Variance of a Binomial random variable
p = 0.5
n = 10
rv = stats.binom(n,p)
M, V = rv.stats(moments='mv')
print("Mean:",M)
print("Variance:",V)

# Code to Generate binomial CDF
p = 0.5
n = 10
rv = stats.binom(n,p)
x = np.arange(11)
F = rv.cdf(x)
plt.plot(x, F, 'bo', ms=10);
plt.vlines(x, 0, F, colors='b', lw=5, alpha=0.5)
plt.title('Binomial' + '\n' + 'CDF')
plt.xlabel('State')
plt.ylabel('Probability')
plt.show()
```

- Here we start again by importing the necessary libraries: numpy, matplotlib.pyplot, and scipy.stats.

- Generating Binomial PMF:

set the parameters n and p to define a binomial distribution. then create a binomial random variable object rv using scipy.stats.binom and generates an array x representing the possible values of the random variable. then compute the probability mass function (PMF) of the binomial distribution using the pmf method of the rv object and plots it using plt.plot. The plt.vlines function is used to add vertical lines to the plot, representing the probability at each value. Then setting plot title and axis labels .

- Generating Binomial Random Variables:

The code generates 5000 random variables from a binomial distribution with parameters n and p using np.random.binomial. It stores the random variables in the array X. then plots a histogram of the random variables using plt.hist. The bins='auto' argument automatically determines the number of bins for the histogram. The plot title and axis labels are set.

- Computing Mean and Variance:

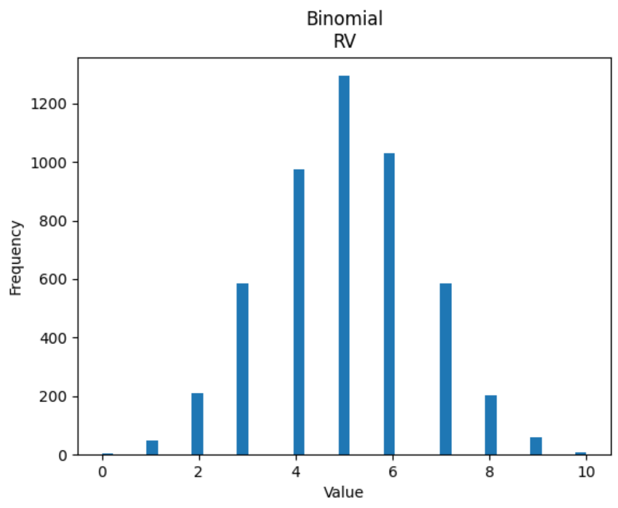
we calculate the mean and variance of a binomial random variable with parameters n and p using the stats.binom object. The stats.binom.stats function with moments='mv' returns the mean and variance. The mean is stored in the variable M and the variance is stored in V. Then they are printed.

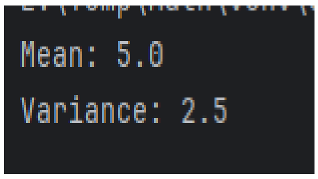
- Generating Binomial CDF:

we set the parameters n and p to define a binomial distribution. then create a binomial random variable object rv using scipy.stats.binom and generate an array x representing the possible values of the random variable. then compute the cumulative distribution function (CDF) of the binomial distribution using the cdf method of the rv object and plots it using plt.plot. The plt.vlines function is used to add vertical lines to the plot, representing the cumulative probability at each value. The plot title and axis labels are set.

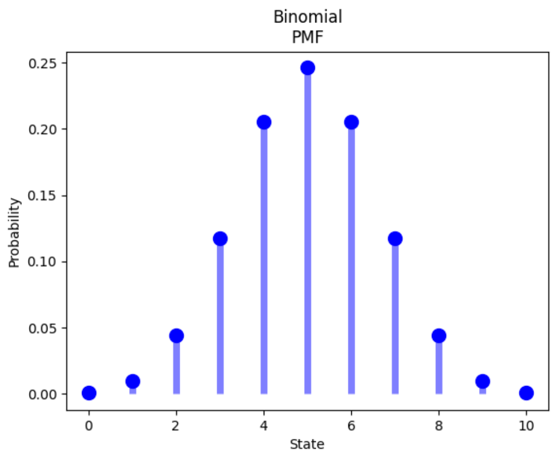
Overall, this code demonstrates the generation and visualization of a binomial distribution, including the PMF, histogram of random variables, mean and variance computation, and CDF.

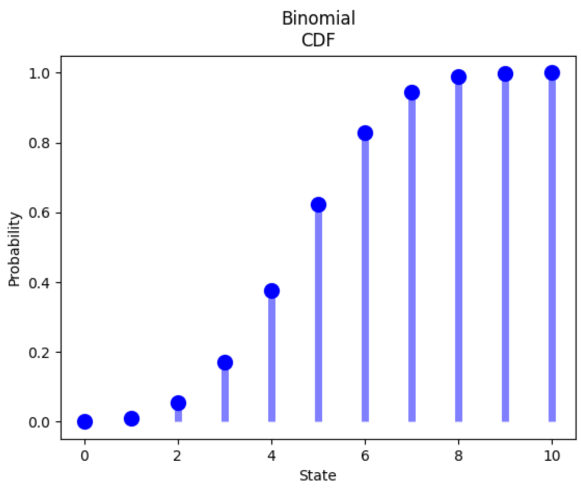
Run of code





```
Mean: 5.0
Variance: 2.5
```





➢ ***Real-Life Example of the Binomial***

Number of River Overflows Park systems use the binomial distribution to model the probability that rivers overflow a certain number of times each year due to excessive rain.

For example, suppose it is known that a given river overflows during 5% of all storms. If there are 20 storms in a given year, we can use a Binomial Distribution Calculator to find the probability that the river overflows a certain number of times:

- P (X = 0 overflows) = 0.35849
- P (X = 1 overflow) = 0.37735
- P (X = 2 overflows) = 0.18868

And so on.

This gives the parks departments an idea of how many times they may need to prepare for overflows throughout the

Poisson distribution

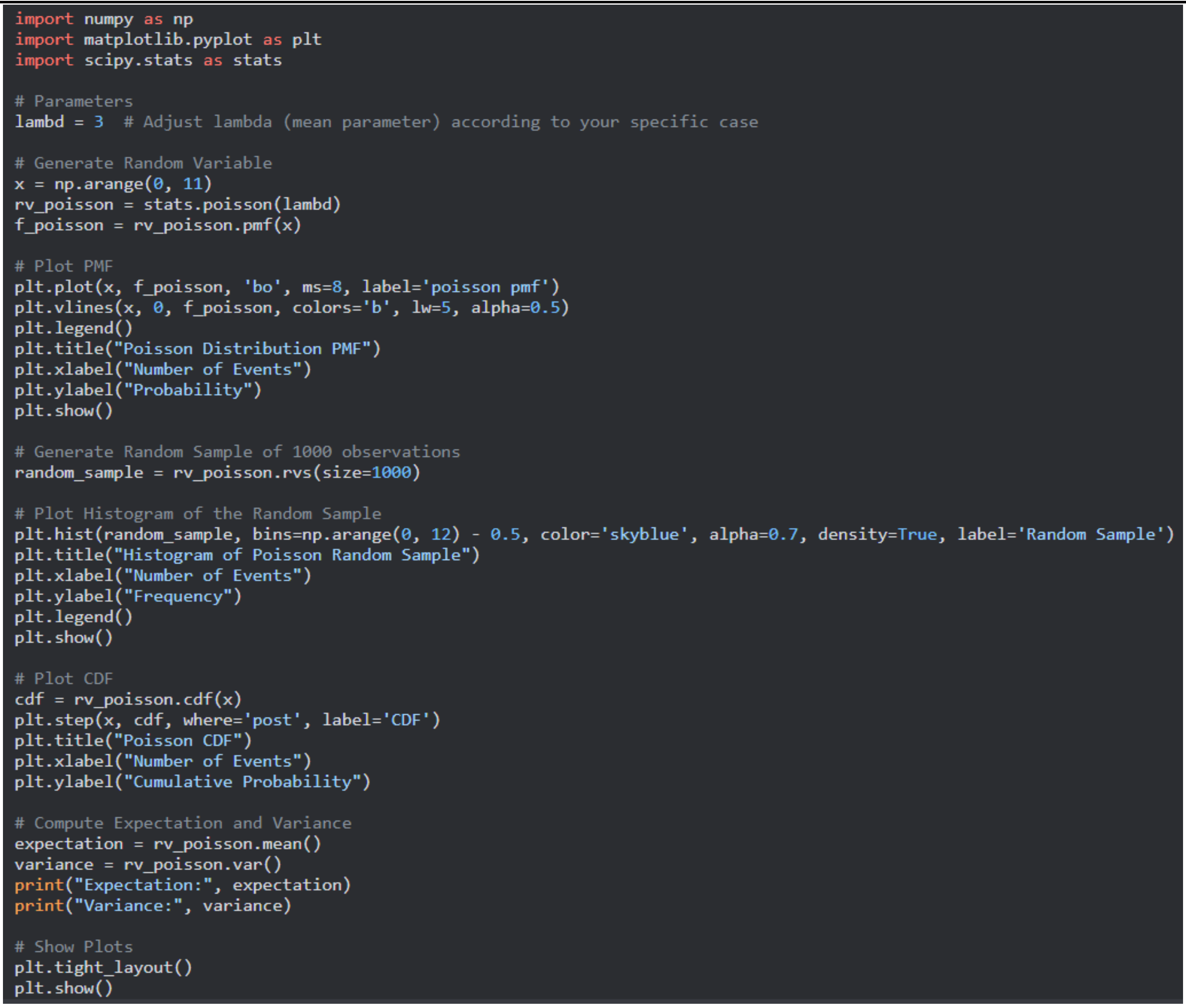
The Poisson distribution is a discrete probability distribution that describes the probability of a given number of events occurring in a fixed interval of time or space when the events happen at a constant average rate and are independent of each other. It is often used to model the occurrence of rare events or the number of events within a specific time period.

The formula for the Poisson distribution function is given by:

$f(x) = (e^{-\lambda}\lambda^{x})/x!$

Where:

- e is the base of the logarithm.
- x is a Poisson random variable.
- λ is an average rate of value.



```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

# Parameters
lambd = 3  # Adjust lambda (mean parameter) according to your specific case

# Generate Random Variable
x = np.arange(0, 11)
rv_poisson = stats.poisson(lambd)
f_poisson = rv_poisson.pmf(x)

# Plot PMF
plt.plot(x, f_poisson, 'bo', ms=8, label='poisson pmf')
plt.vlines(x, 0, f_poisson, colors='b', lw=5, alpha=0.5)
plt.legend()
plt.title("Poisson Distribution PMF")
plt.xlabel("Number of Events")
plt.ylabel("Probability")
plt.show()

# Generate Random Sample of 1000 observations
random_sample = rv_poisson.rvs(size=1000)

# Plot Histogram of the Random Sample
plt.hist(random_sample, bins=np.arange(0, 12) - 0.5, color='skyblue', alpha=0.7, density=True, label='Random Sample')
plt.title("Histogram of Poisson Random Sample")
plt.xlabel("Number of Events")
plt.ylabel("Frequency")
plt.legend()
plt.show()

# Plot CDF
cdf = rv_poisson.cdf(x)
plt.step(x, cdf, where='post', label='CDF')
plt.title("Poisson CDF")
plt.xlabel("Number of Events")
plt.ylabel("Cumulative Probability")

# Compute Expectation and Variance
expectation = rv_poisson.mean()
variance = rv_poisson.var()
print("Expectation:", expectation)
print("Variance:", variance)

# Show Plots
plt.tight_layout()
plt.show()
```

- First begin by importing the libraries

- Define Parameters:
  We set parameter called lambd to a value of 3. This parameter represents the mean (lambda) of the Poisson distribution.

- Generate Random Variable:
  generate an array x using np.arange to represent the possible values of the random variable. create a Poisson random variable object rv_poisson using scipy.stats.poisson and sets the lambda parameter to the value of lambd. then compute the probability mass function (PMF) of the Poisson distribution at each value in x using the pmf method of the rv_poisson object and stores it in the variable f_poisson.

- Plot PMF:
  we plot the PMF of the Poisson distribution using plt.plot. use blue dots ('bo') as markers for the plot and sets the marker size to 8. also add vertical lines using plt.vlines to represent the probability at each value. Then setting labels.

- Generate Random Sample:
  code generates a random sample of 1000 observations from the Poisson distribution using the rvs method of the rv_poisson object and store the sample in variable called random_sample.

- Plotting Histogram of the Random Sample:
  we plot a histogram of the random sample using plt.hist. specify the bin edges using np.arange(0, 12) - 0.5 to create bins with a width of 1.

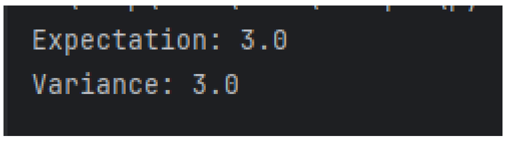
- Plot CDF:
  we compute the cumulative distribution function (CDF) of the Poisson distribution at each value in x using the cdf method of the rv_poisson object. then plot the CDF using plt.step to create a step plot. The where='post' argument specifies that the steps should be placed after the x-values.

- Compute Expectation and Variance:
  we compute the expectation (mean) and variance of the Poisson distribution using the mean and var methods of the rv_poisson object, respectively. The values are stored in the variables expectation and variance and then printed.

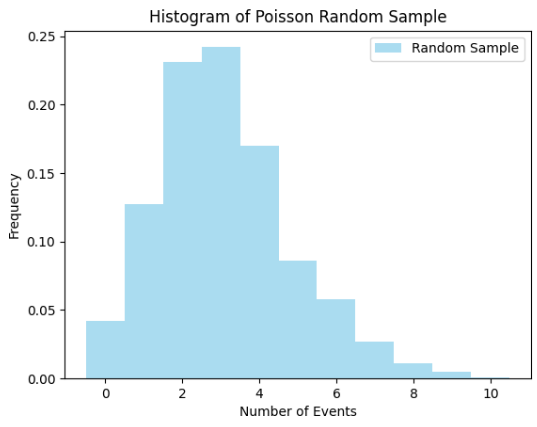
- Showing Plots:
  The code calls plt.tight_layout() to adjust the spacing between subplots and plt.show() to display the plots on the screen.

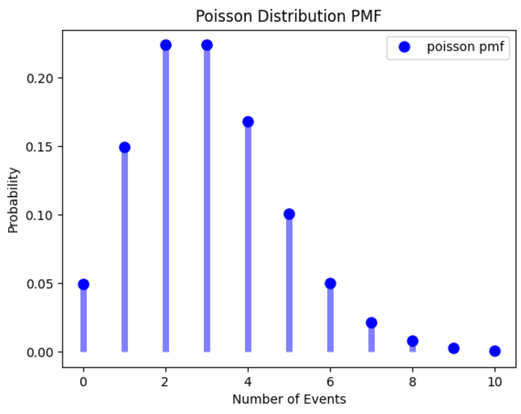
  Overall, this code generates and visualizes a Poisson distribution, including the PMF, histogram of a random sample, CDF, and computes the expectation and variance.

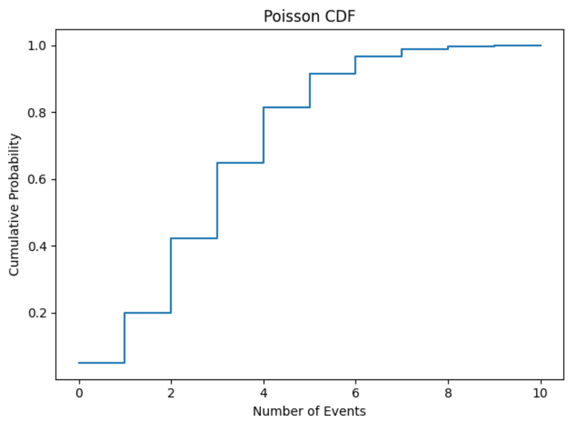
Run of code



```
Expectation: 3.0
Variance: 3.0
```







➢ ***Life application: Number of Website Visitors per Hour:***

Website hosting companies use the Poisson distribution to model the number of expected visitors per

hour that websites will receive.

For example, suppose a given website receives an average of 20 visitors per hour. We can use the Poisson distribution calculator to find the probability that the website receives more than a certain number of visitors in a given hour:

 P (X > 25 visitors) = 0.11218

 P (X > 30 visitors) = 0.01347

 P (X > 35 visitors) = 0.00080

And so on.

This gives hosting companies an idea of how much bandwidth to provide to different websites to ensure

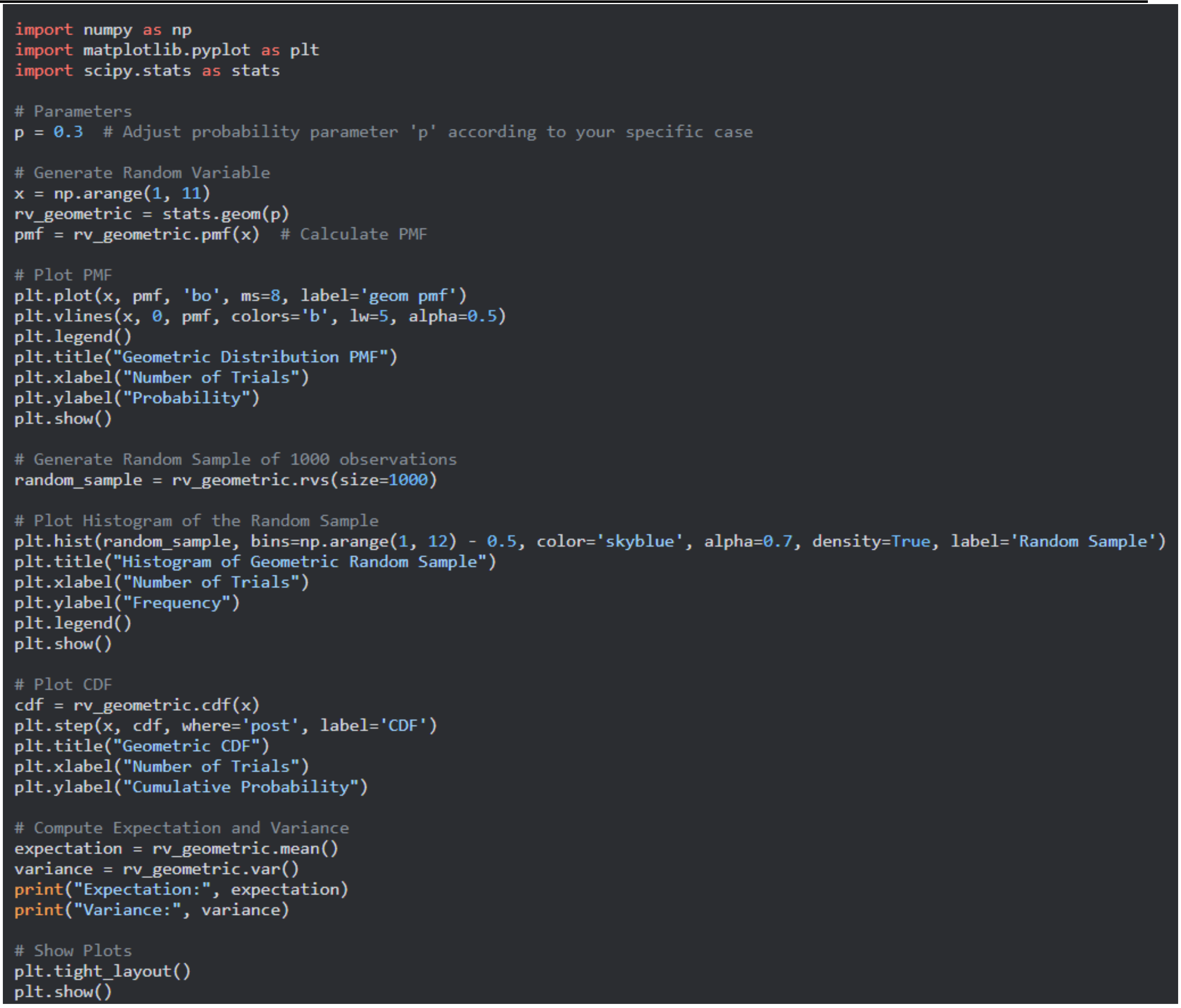
that they'll be able to handle a certain number of visitors each hour.

Geometric distribution

The mean of geometric distribution is also the expected value of the geometric distribution. The expected value of a random variable, X, can be defined as the weighted average of all values of X. The formula for the mean of a geometric distribution is given as follows: E[X] = 1 / p

$$P(X=x) = (1-p)^{x-1}p$$

$$P(X\leq x) = 1-(1-p)^{x}$$



```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

# Parameters
p = 0.3  # Adjust probability parameter 'p' according to your specific case

# Generate Random Variable
x = np.arange(1, 11)
rv_geometric = stats.geom(p)
pmf = rv_geometric.pmf(x)  # Calculate PMF

# Plot PMF
plt.plot(x, pmf, 'bo', ms=8, label='geom pmf')
plt.vlines(x, 0, pmf, colors='b', lw=5, alpha=0.5)
plt.legend()
plt.title("Geometric Distribution PMF")
plt.xlabel("Number of Trials")
plt.ylabel("Probability")
plt.show()

# Generate Random Sample of 1000 observations
random_sample = rv_geometric.rvs(size=1000)

# Plot Histogram of the Random Sample
plt.hist(random_sample, bins=np.arange(1, 12) - 0.5, color='skyblue', alpha=0.7, density=True, label='Random Sample')
plt.title("Histogram of Geometric Random Sample")
plt.xlabel("Number of Trials")
plt.ylabel("Frequency")
plt.legend()
plt.show()

# Plot CDF
cdf = rv_geometric.cdf(x)
plt.step(x, cdf, where='post', label='CDF')
plt.title("Geometric CDF")
plt.xlabel("Number of Trials")
plt.ylabel("Cumulative Probability")

# Compute Expectation and Variance
expectation = rv_geometric.mean()
variance = rv_geometric.var()
print("Expectation:", expectation)
print("Variance:", variance)

# Show Plots
plt.tight_layout()
plt.show()
```

- First Importing libraries:

imports the necessary libraries: NumPy for numerical operations, Matplotlib for plotting, and SciPy for statistical functions.

- Setting the parameter:

we set the parameter 'p' to a specific value. The geometric distribution represents the probability of the first success occurring on the k-th trial, and 'p' represents the probability of success on each trial.

- Generating the random variable 'x':

This code generates an array 'x' containing values from 3 to 10. These values represent the possible number of trials in the geometric distribution.

- Calculating the PMF (Probability Mass Function):

Here, we create a geometric distribution object 'rv_geometric' using the 'geom' function from SciPy's stats module, passing the parameter 'p'. Then, calculate the PMF for the values in 'x' using the 'pmf' method of the distribution object.

- Plotting the PMF:

code creates a plot of the PMF using Matplotlib. The 'plt.plot' function plots the points, while 'plt.vlines' draws vertical lines from the x-axis to the PMF values. The other 'plt' functions set the labels, title, and legend for the plot, and 'plt.show()' displays the plot.

- Generating a random sample:

code generates a random sample of 1000 observations from the geometric distribution using the 'rvs' method of the distribution object.

- Plotting the histogram of the random sample:

We create a histogram of the random sample using 'plt.hist' function. The 'bins' parameter specifies the bin edges, and the other 'plt' functions set the labels and title for the plot.

- Plotting the CDF (Cumulative Distribution Function):

we calculate the CDF for the values in 'x' using the 'cdf' method of the distribution object and plots it using 'plt.step'. The other 'plt' functions set the labels and title for the plot.

➢ Computing the expectation and variance:

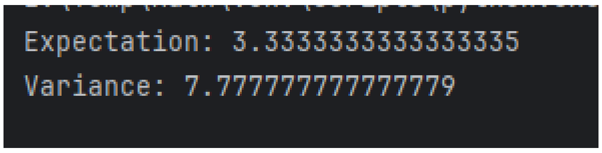
Here we calculate the expectation (mean) and variance of the geometric distribution using the 'mean' and 'var' methods of the distribution object, respectively. Then print it.

➢ Showing the plots:

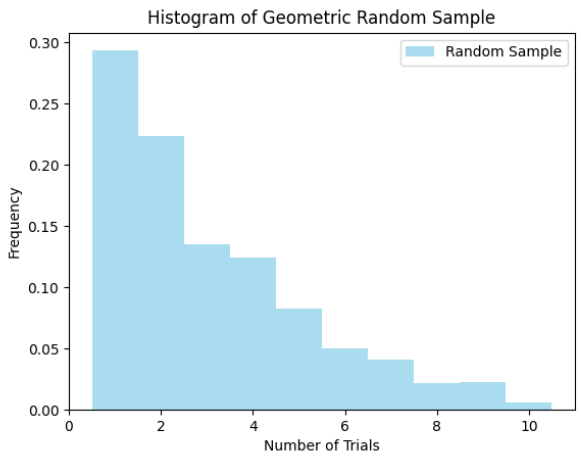
Here we adjust the layout of the plots and display them on the screen.

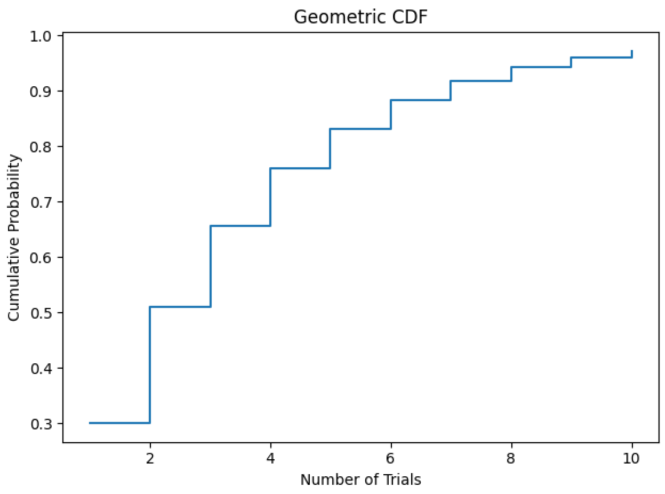
Overall, this code demonstrates how to generate data from a geometric distribution, calculate the PMF, plot the PMF, generate a random sample, plot a histogram of the sample, plot the CDF, and compute the expectation and variance.

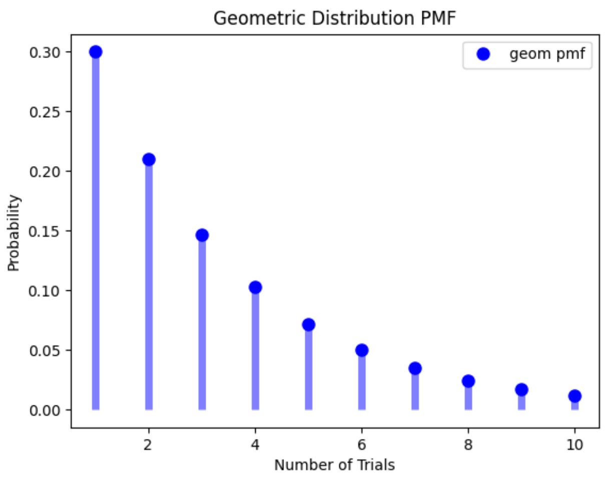
Run of code



```
Expectation: 3.3333333333333335
Variance: 7.777777777777779
```







Discrete Uniform Distribution

A discrete uniform probability distribution is a distribution with constant probability, meaning that a finite number of values are equally likely to be observed.

This type of distribution is defined by two parameters:
1. a – the minimum
2. b – the maximum

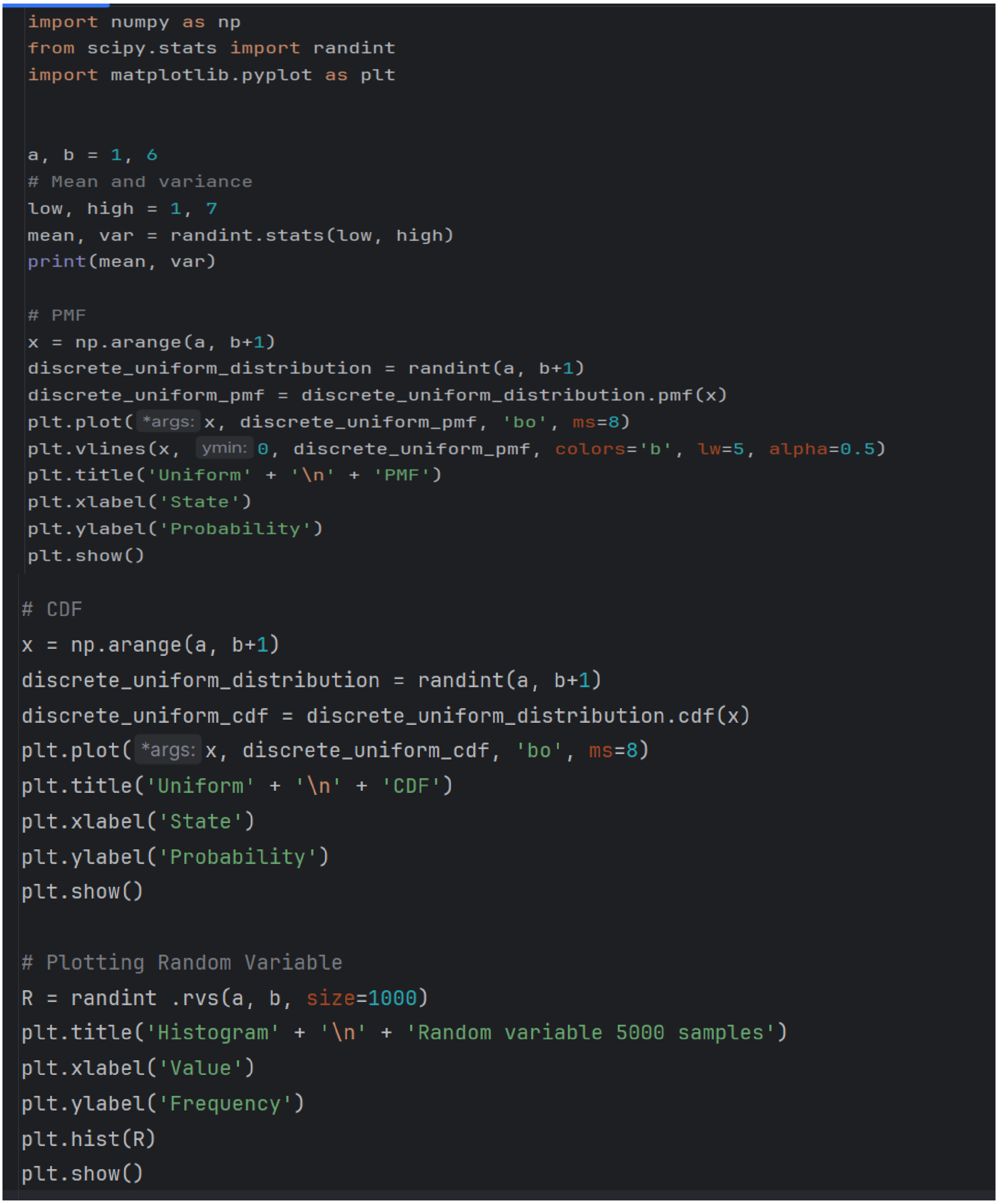
For any x∈ [a, b], the PMF (probability mass function) of a discrete uniform distribution is given by f(x)=1/(b−a+1) =1/n

And for any x∈ [a, b], the CDF (cumulative distribution function) of a discrete uniform distribution is given by: F(x)=P(X≤x) =(x−a+1)/(b−a+1) =(x−a+1)/n

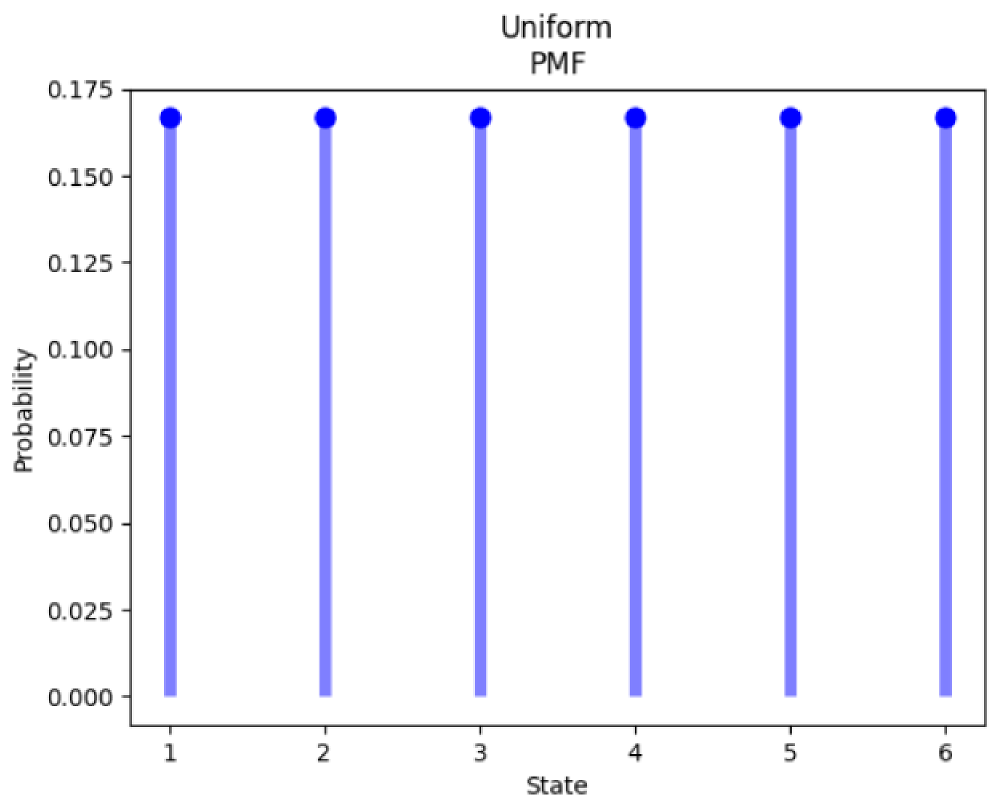
Mean: $\frac{a+b}{2}$

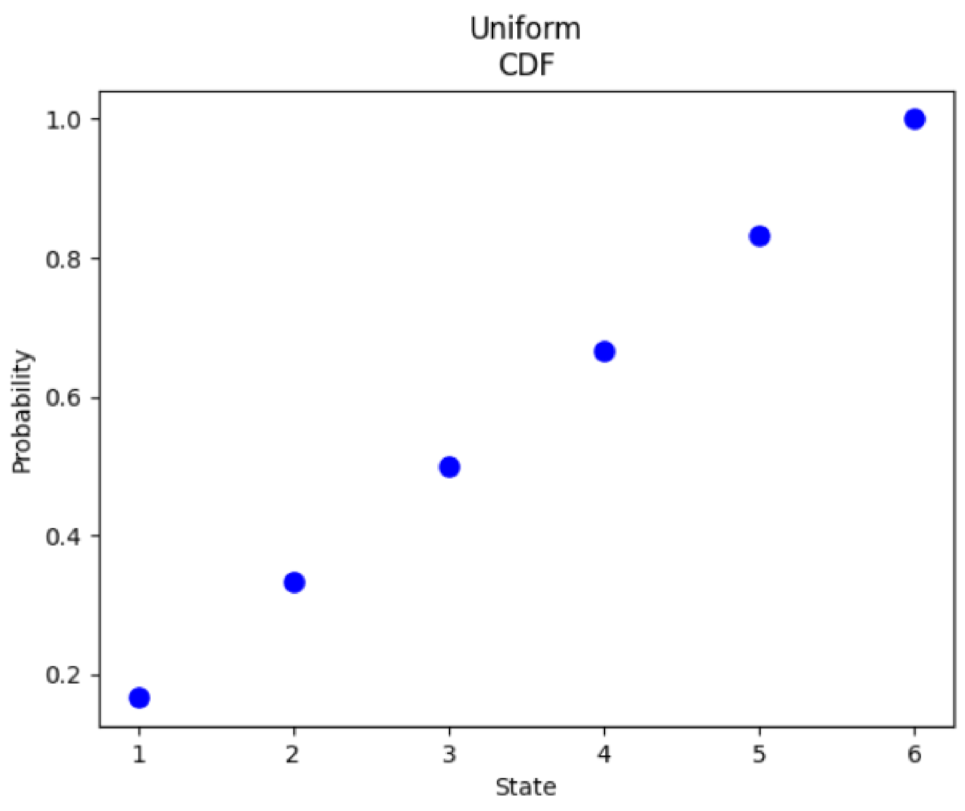
Variance: $\frac{(b-a)^2}{12}$

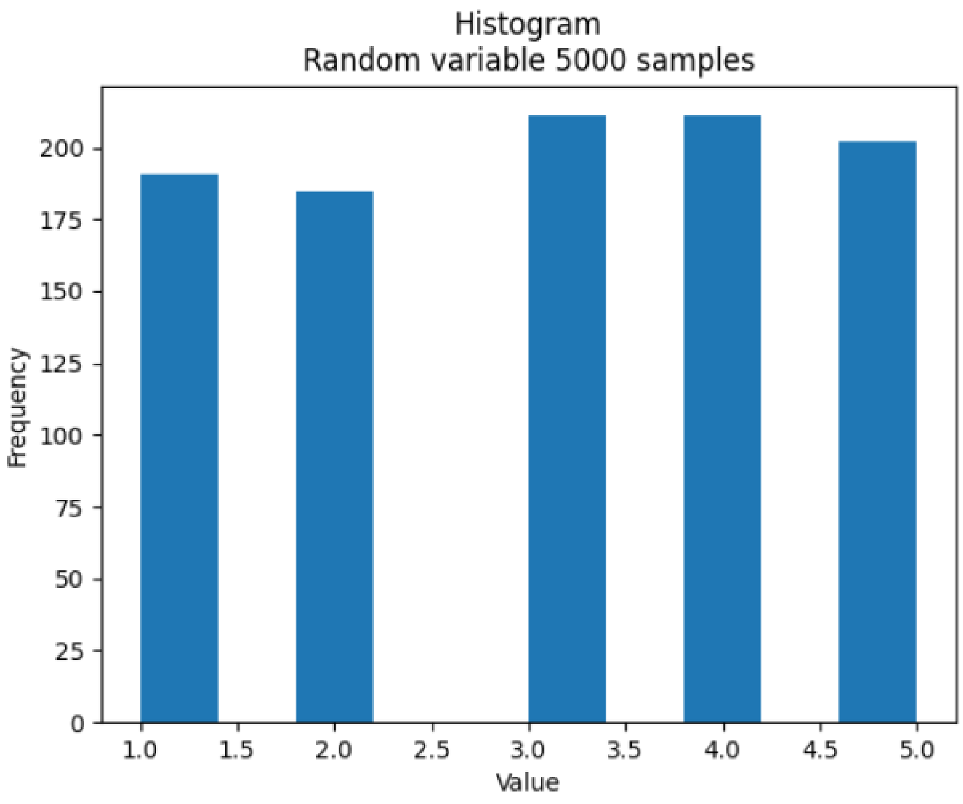
Run of Code:

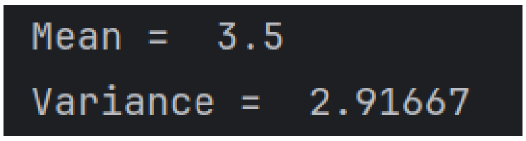


```
import numpy as np
from scipy.stats import randint
import matplotlib.pyplot as plt


a, b = 1, 6
# Mean and variance
low, high = 1, 7
mean, var = randint.stats(low, high)
print(mean, var)

# PMF
x = np.arange(a, b+1)
discrete_uniform_distribution = randint(a, b+1)
discrete_uniform_pmf = discrete_uniform_distribution.pmf(x)
plt.plot( *args: x, discrete_uniform_pmf, 'bo', ms=8)
plt.vlines(x,  ymin: 0, discrete_uniform_pmf, colors='b', lw=5, alpha=0.5)
plt.title('Uniform' + '\n' + 'PMF')
plt.xlabel('State')
plt.ylabel('Probability')
plt.show()

# CDF
x = np.arange(a, b+1)
discrete_uniform_distribution = randint(a, b+1)
discrete_uniform_cdf = discrete_uniform_distribution.cdf(x)
plt.plot( *args: x, discrete_uniform_cdf, 'bo', ms=8)
plt.title('Uniform' + '\n' + 'CDF')
plt.xlabel('State')
plt.ylabel('Probability')
plt.show()

# Plotting Random Variable
R = randint .rvs(a, b, size=1000)
plt.title('Histogram' + '\n' + 'Random variable 5000 samples')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.hist(R)
plt.show()
```


Uniform
PMF
Probability
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
1
2
3
4
5
6
State


Uniform
CDF
Probability
1.0
0.8
0.6
0.4
0.2
1
2
3
4
5
6
State





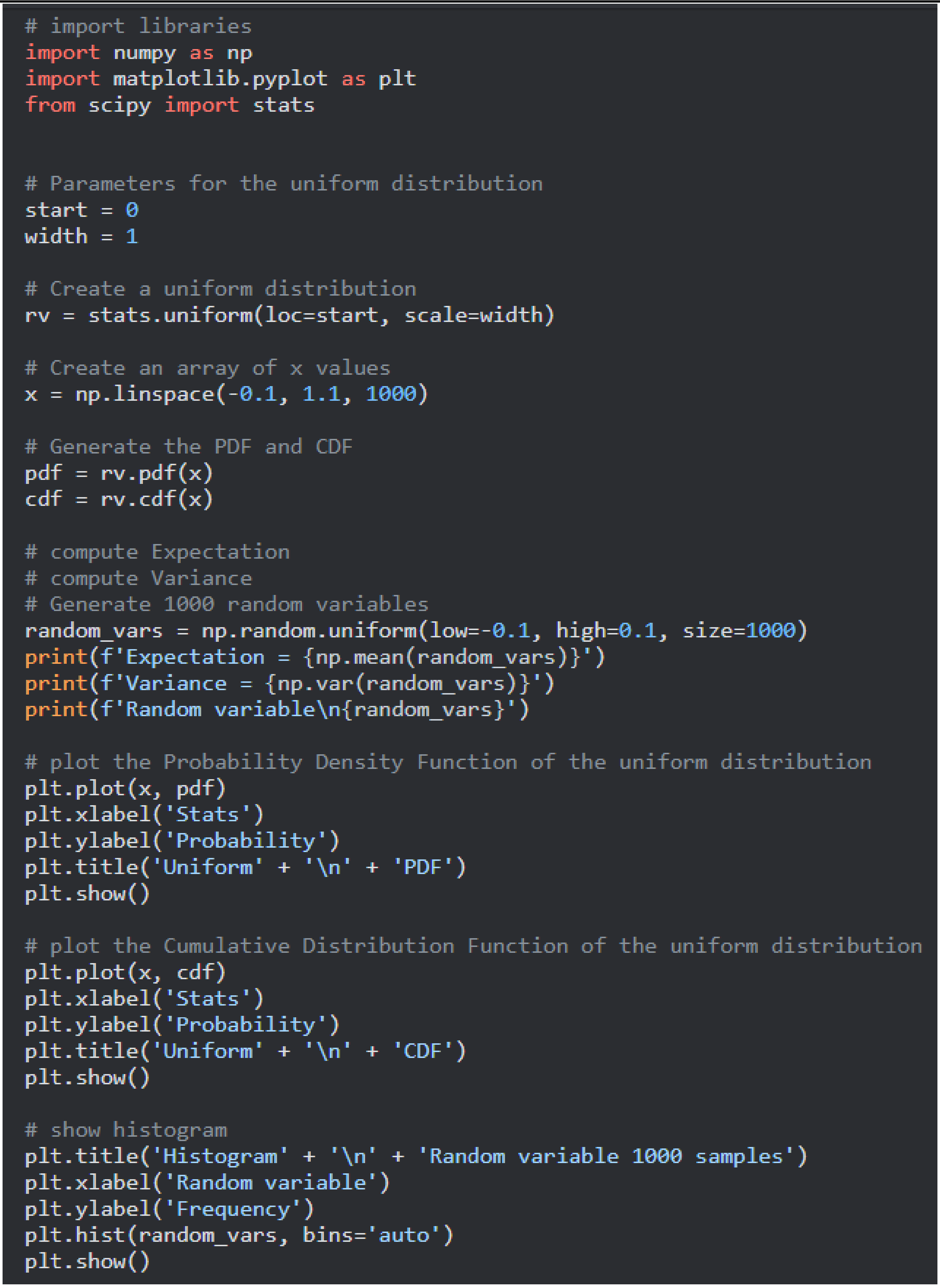
```
Mean =  3.5
Variance =  2.91667
```

Continuous Random Variables

Uniform Distribution

Mean : $\int_{-\infty}^{\infty} x.f(x)dx$

Variance: $E(x^2)-[E(x)]^2$



```python
# import libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats


# Parameters for the uniform distribution
start = 0
width = 1

# Create a uniform distribution
rv = stats.uniform(loc=start, scale=width)

# Create an array of x values
x = np.linspace(-0.1, 1.1, 1000)

# Generate the PDF and CDF
pdf = rv.pdf(x)
cdf = rv.cdf(x)

# compute Expectation
# compute Variance
# Generate 1000 random variables
random_vars = np.random.uniform(low=-0.1, high=0.1, size=1000)
print(f'Expectation = {np.mean(random_vars)}')
print(f'Variance = {np.var(random_vars)}')
print(f'Random variable\n{random_vars}')

# plot the Probability Density Function of the uniform distribution
plt.plot(x, pdf)
plt.xlabel('Stats')
plt.ylabel('Probability')
plt.title('Uniform' + '\n' + 'PDF')
plt.show()

# plot the Cumulative Distribution Function of the uniform distribution
plt.plot(x, cdf)
plt.xlabel('Stats')
plt.ylabel('Probability')
plt.title('Uniform' + '\n' + 'CDF')
plt.show()

# show histogram
plt.title('Histogram' + '\n' + 'Random variable 1000 samples')
plt.xlabel('Random variable')
plt.ylabel('Frequency')
plt.hist(random_vars, bins='auto')
plt.show()
```

- First import the necessary libraries (numpy) for numerical computations, (matplotlib.pyplot) for data visualization , (scipy.stats) for statistical functions .

- Define Parameters for the Uniform Distribution:

These variables define the parameters for the uniform distribution. start represents the starting point of the distribution, and width represents the range of the distribution.

- Creating a Uniform Distribution Random Variable:

create a random variable object (rv) representing a uniform distribution using the stats.uniform function from the scipy library. The loc parameter represents the starting point of the distribution (start), and the scale parameter represents the width of the distribution (width). Then creating an Array of x Values that generate an array of 1000 equally spaced values between -0.1 and 1.1 using the np.linspace function from the numpy library. These values will be used to evaluate the probability density function (PDF) and cumulative distribution function (CDF) of the uniform distribution.

- Generating the Probability Density Function (PDF) and Cumulative Distribution Function (CDF):

compute the PDF and CDF of the uniform distribution. The pdf variable stores the PDF values evaluated at each value in x, and the cdf variable stores the CDF values evaluated at each value in x.

- Generating Random Variables:

We generate an array of 1000 random variables from the uniform distribution using the np.random.uniform function from the numpy library. The low and high parameters represent the range of values for the random variables.

- Computing Expectation and Variance:

we compute the expectation (mean) and variance of uniform distribution using the mean and var methods of the random_vars object then print it.

- Plotting the Probability Density Function (PDF):

we create a line plot of the PDF values (pdf) against the corresponding x values. Then set the labels and title using plt.xlabel, plt.ylabel, and plt.title. and use, plt.show() to display the plot.

➢ Plotting the Cumulative Distribution Function (CDF):

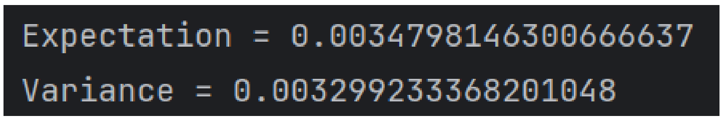
line plot of the CDF values (cdf) against the corresponding x values. The plt.xlabel, plt.ylabel, and plt.title functions set the labels and title for the plot. Finally, plt.show() displays the plot.

➢ Plotting the Histogram:

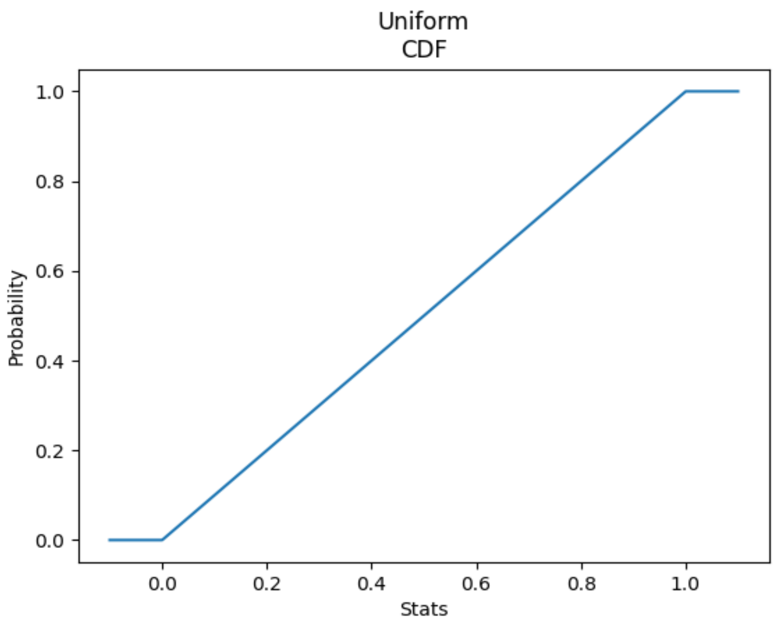
We use plt.hist function to generate the histogram using the random variable data and the bins='auto' parameter automatically determines the number of bins. Then set the labels and title and finally use plt.show() to display the plot.

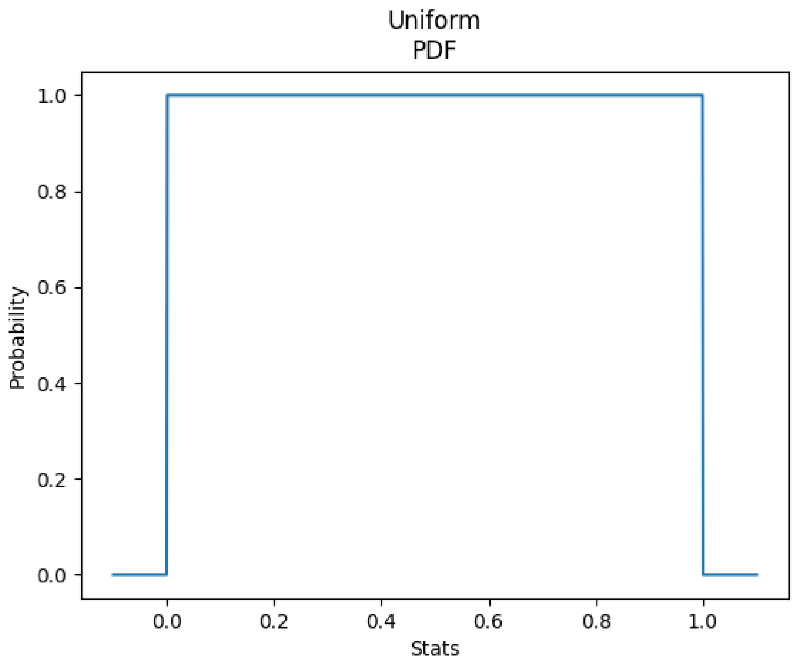
Overall, The code generates and analyzes a uniform distribution, including plotting the PDF, CDF, and histogram.

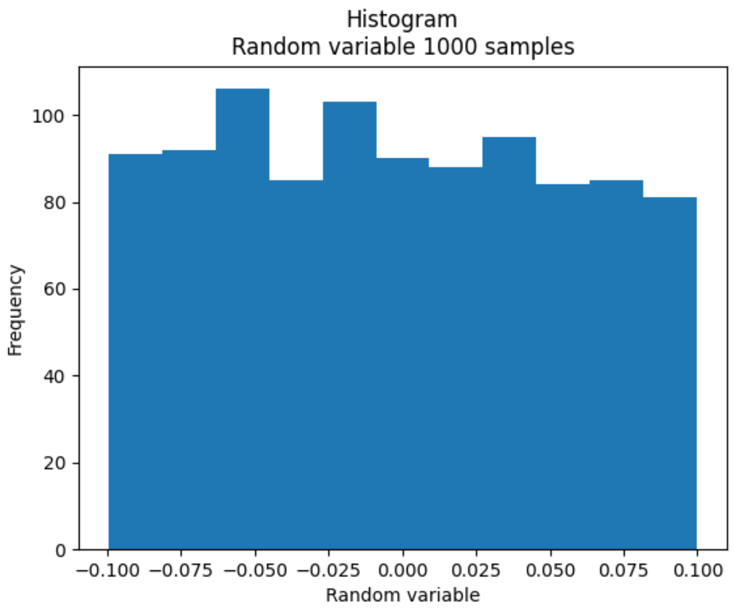
Run of code



```
Expectation = 0.003479814630066637
Variance = 0.003299233368201048
```



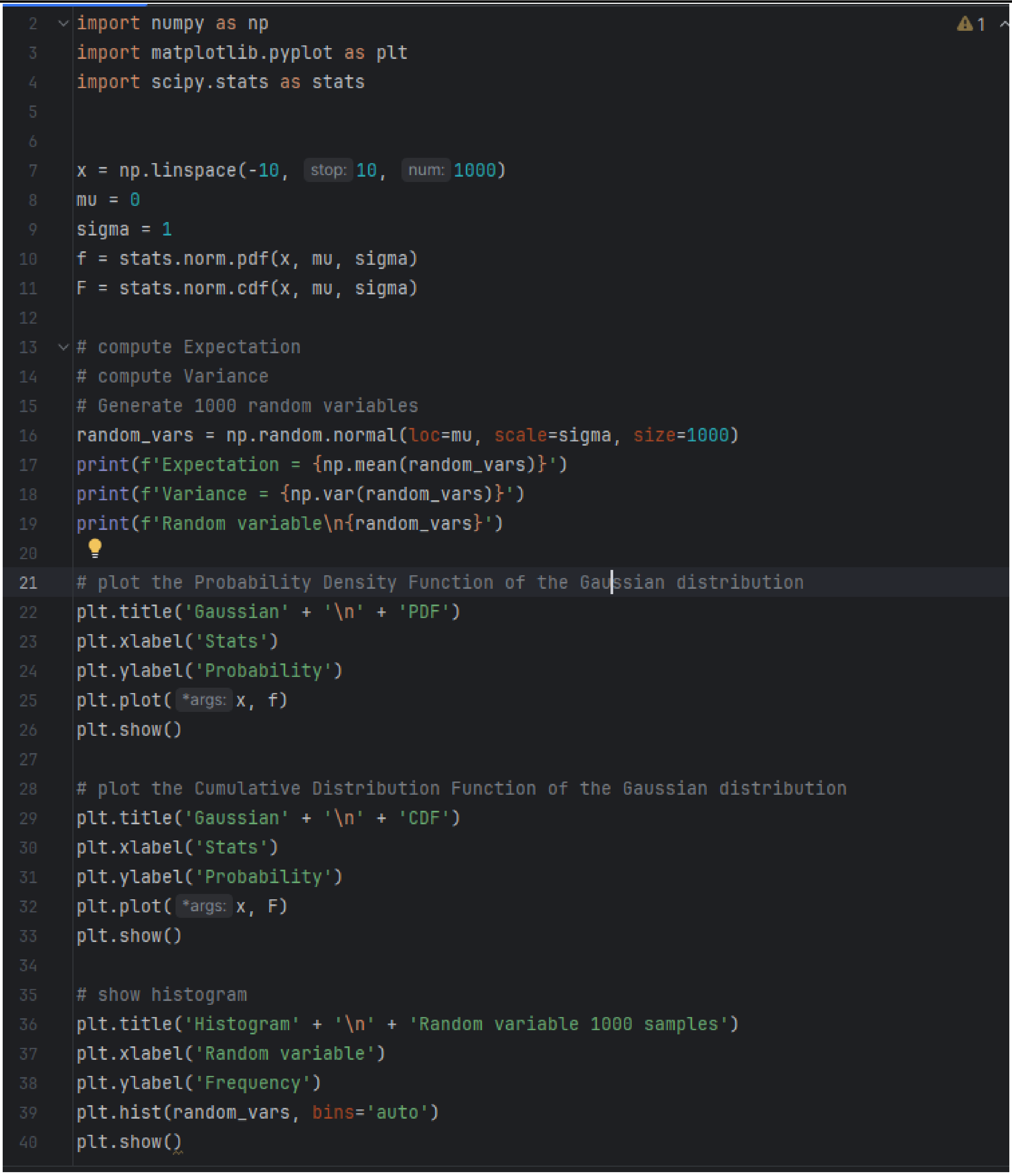

Uniform
PDF
Probability
Stats
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Histogram
Random variable 1000 samples
Frequency
Random variable
100
80
60
40
20
0
−0.100
−0.075
−0.050
−0.025
0.000
0.025
0.050
0.075
0.100

Gaussian Distribution

In the study of random variables, the Gaussian random variable is clearly the most commonly used and of most importance. As we will see later in the text, many physical phenomena can be modeled as Gaussian random variables, including the thermal noise encountered in electronic circuits.

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



```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats


x = np.linspace(-10,  stop: 10,  num: 1000)
mu = 0
sigma = 1
f = stats.norm.pdf(x, mu, sigma)
F = stats.norm.cdf(x, mu, sigma)

# compute Expectation
# compute Variance
# Generate 1000 random variables
random_vars = np.random.normal(loc=mu, scale=sigma, size=1000)
print(f'Expectation = {np.mean(random_vars)}')
print(f'Variance = {np.var(random_vars)}')
print(f'Random variable\n{random_vars}')

# plot the Probability Density Function of the Gaussian distribution
plt.title('Gaussian' + '\n' + 'PDF')
plt.xlabel('Stats')
plt.ylabel('Probability')
plt.plot( *args: x, f)
plt.show()

# plot the Cumulative Distribution Function of the Gaussian distribution
plt.title('Gaussian' + '\n' + 'CDF')
plt.xlabel('Stats')
plt.ylabel('Probability')
plt.plot( *args: x, F)
plt.show()

# show histogram
plt.title('Histogram' + '\n' + 'Random variable 1000 samples')
plt.xlabel('Random variable')
plt.ylabel('Frequency')
plt.hist(random_vars, bins='auto')
plt.show()
```

1. Probability Density Function (PDF) and Cumulative Distribution Function (CDF)

- ➢ Import Libraries:

NumPy (‘ import numpy as np ’): Explain that NumPy is a powerful library for numerical operations, and here it's used for creating arrays and performing mathematical operations.

Matplotlib (‘ import matplotlib.pyplot as plt ’): Describe Matplotlib as a visualization library. It's employed here for plotting the probability density function (PDF), cumulative distribution function (CDF), and histograms.

SciPy Stats (‘ import scipy.stats as stats ’): Introduce SciPy Stats as a library for statistical functions. In this code, it provides functions for the normal distribution of PDF and CDF.

- ➢ Generating ‘x’ Values:

‘ numpy.linspace ’: Detail how numpy.linspace is utilized to create an array of x values evenly spaced between -10 and 10, providing a range for the normal distribution.

- ➢ PDF Calculation (`f`) and CDF Calculation (`F`):

‘ stats.norm.pdf ’ and ‘ stats.norm.cdf ’: Elaborate on how these functions compute the probability density and cumulative probability, respectively, for each x value. Emphasize the role of parameters mu (mean) and sigma (standard deviation) in shaping the distribution.

2. Expectation and Variance

- ➢ Computation of Expectation and Variance:

`numpy.mean` and `numpy.var`: Explain that these functions calculate the mean (expectation) and variance of the generated random variables, providing insights into the central tendency and variability of the data.

3. Random Variable Generation

- Generating 1000 Random Variables:

`numpy.random.normal`: Describe how this function generates 1000 random variables following a normal distribution with the specified mean (`mu`) and standard deviation (`sigma`). Highlight the importance of random variables in statistical simulations.

- Visualization and Interpretation:

1. Probability Density Function (PDF) Plot

`matplotlib.pyplot.plot`: Explain how this function is employed to visualize the PDF, representing the likelihood of different values occurring in the normal distribution. Discuss the bell-shaped curve, emphasizing symmetry and concentration around the mean.

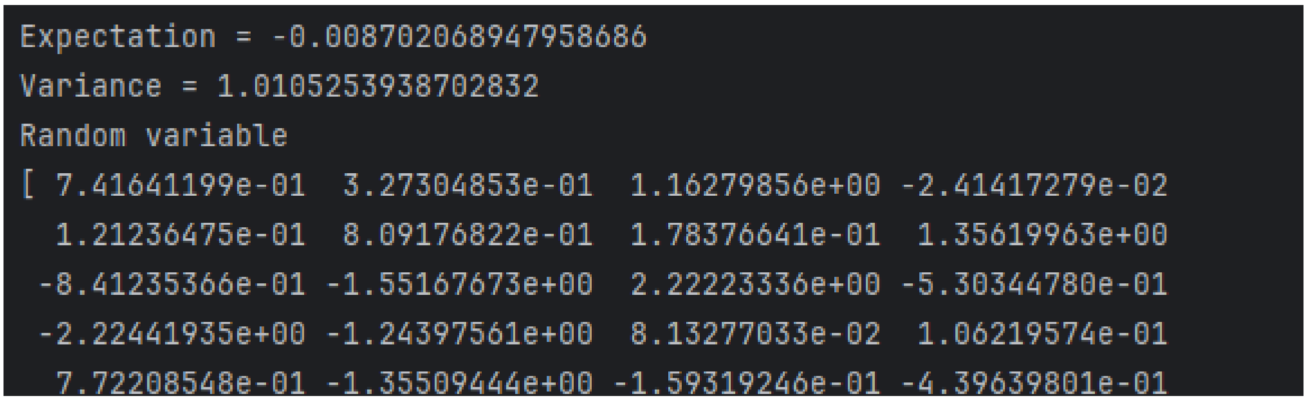
2. Cumulative Distribution Function (CDF) Plot

`matplotlib.pyplot.plot`: Detail the role of this function in illustrating the cumulative probability up to each x point. Discuss the S-shaped curve, explaining the increasing likelihood of values being less than or equal to a specific data point.

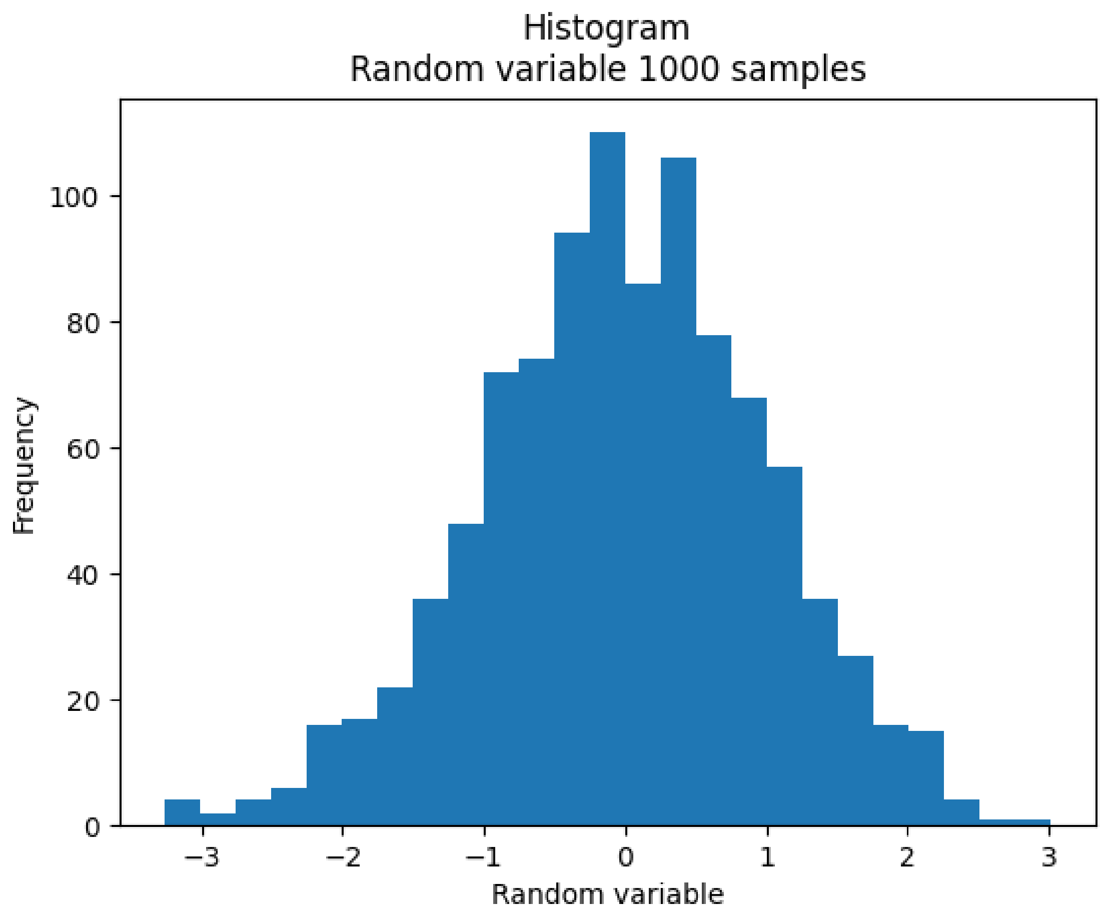
3. Histogram

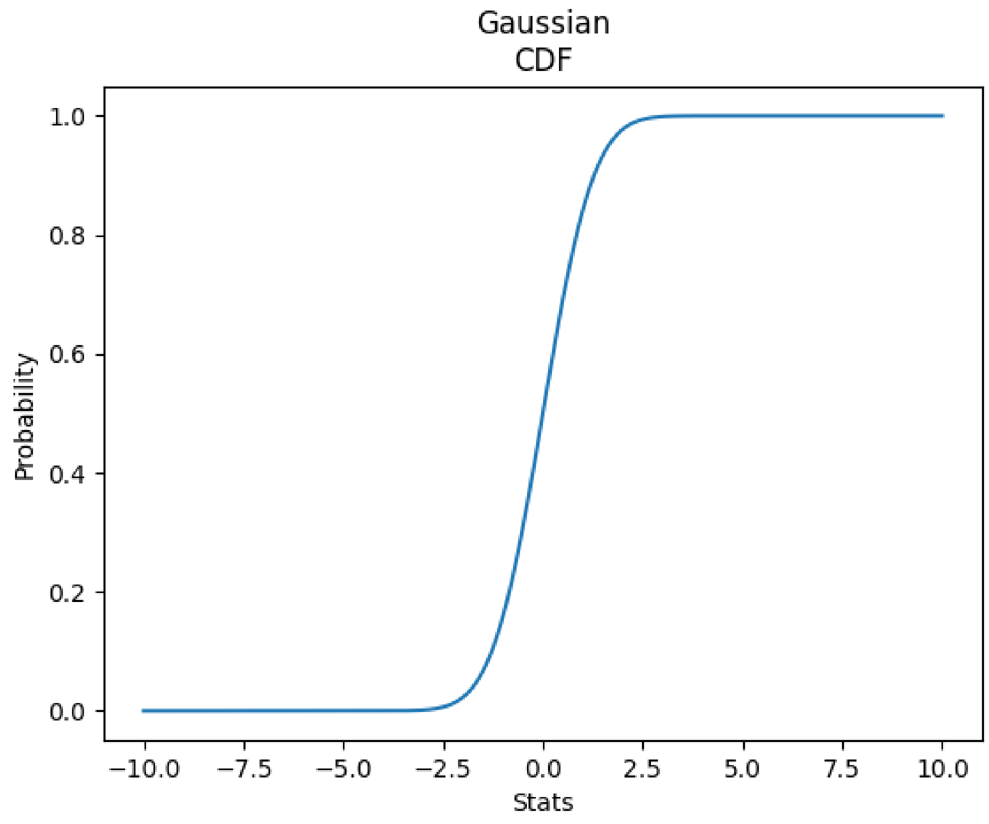
`matplotlib.pyplot.hist`: Provide insights into how this function creates a histogram of the generated random variables, visually approximating the normal distribution. Discuss the importance of choosing an appropriate number of bins for an accurate representation.

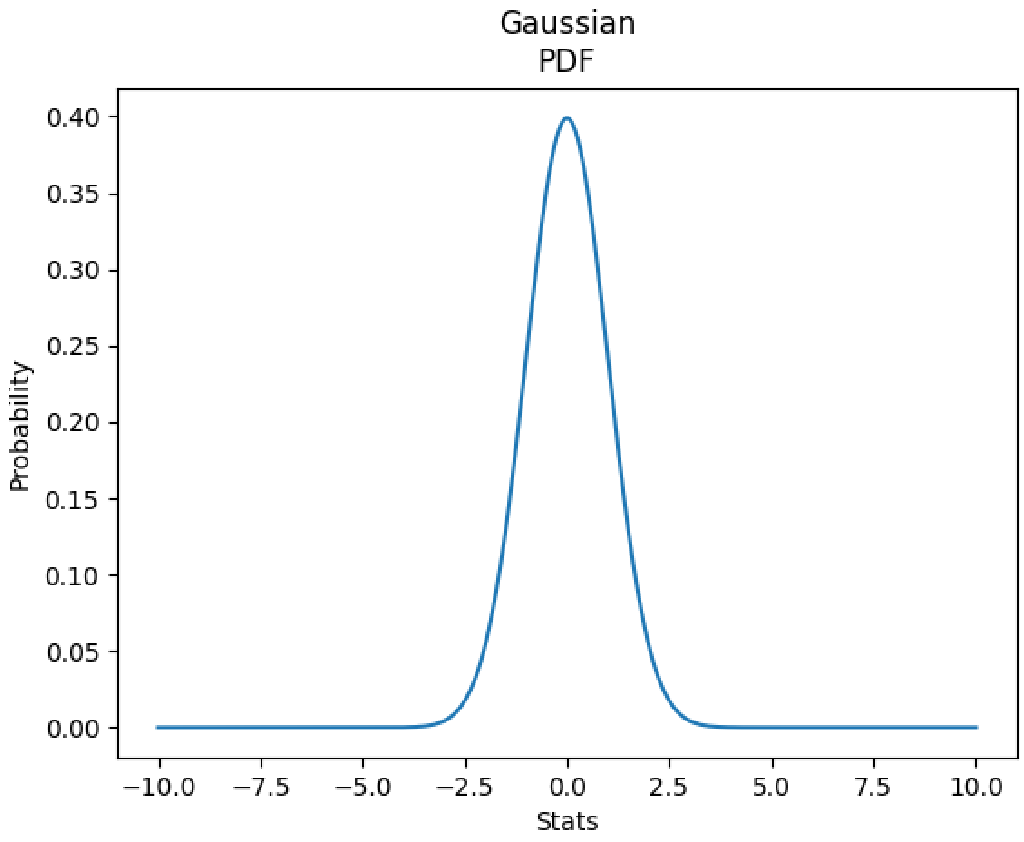
➢ ***Run of code***



```
Expectation = -0.008702068947958686
Variance = 1.0105253938702832
Random variable
[ 7.41641199e-01  3.27304853e-01  1.16279856e+00 -2.41417279e-02
  1.21236475e-01  8.09176822e-01  1.78376641e-01  1.35619963e+00
 -8.41235366e-01 -1.55167673e+00  2.22223336e+00 -5.30344780e-01
 -2.22441935e+00 -1.24397561e+00  8.13277033e-02  1.06219574e-01
  7.72208548e-01 -1.35509444e+00 -1.59319246e-01 -4.39639801e-01
```

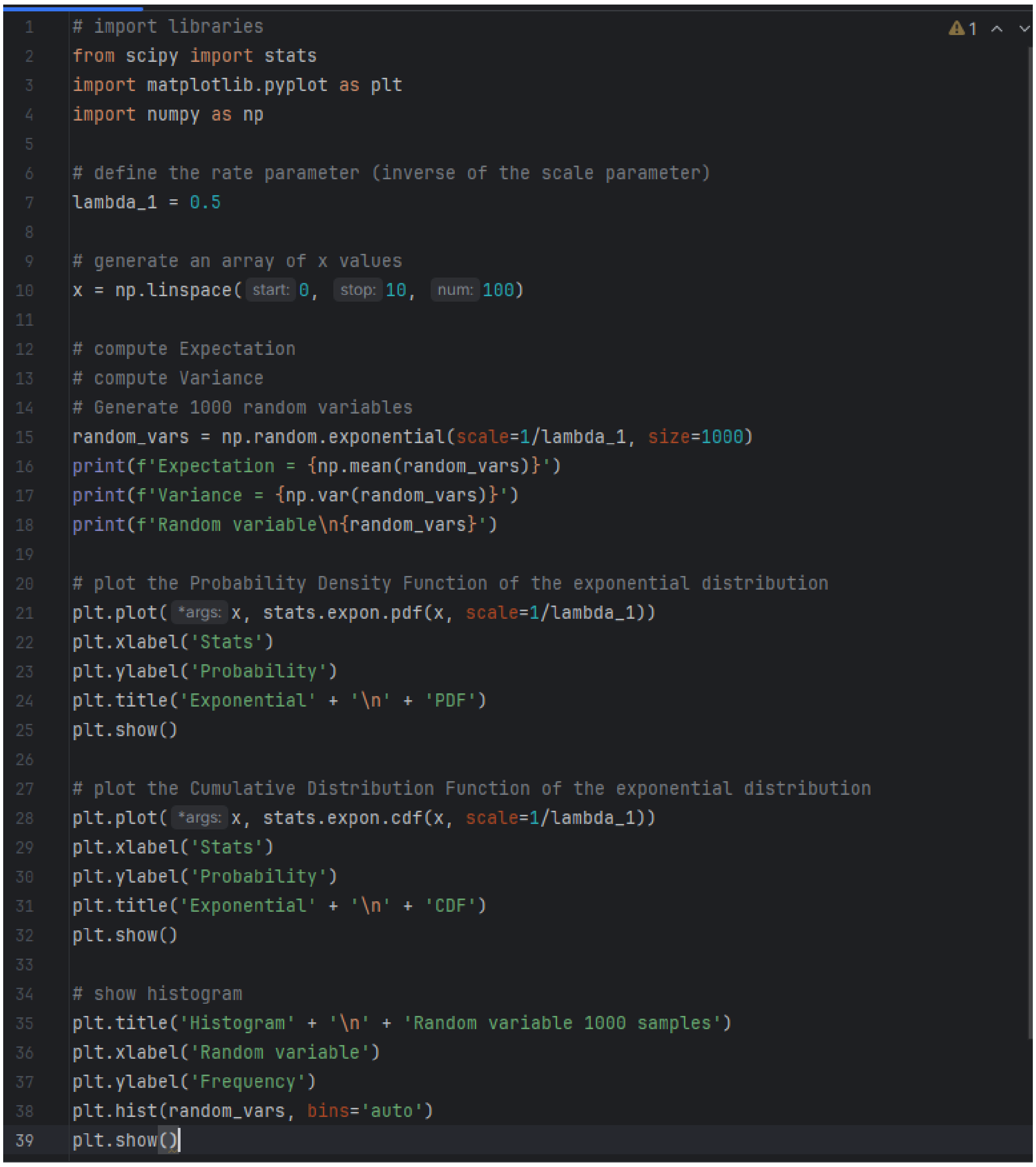



Gaussian
CDF
1.0
0.8
0.6
0.4
0.2
0.0
Probability
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Stats


Gaussian
PDF
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Probability
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Stats

Exponential Distribution

In Probability theory and statistics, the exponential distribution is continuous probability distribution that often concerns the amount of time until some specific event happens. It is a process in which events happen continuously and independently at a constant average rate. Exponential distribution has the key property of being memoryless. The exponential random variable can be either more small values or fewer larger variables. For example, the amount of money spent by the customer on one trip to the supermarket follows an exponential distribution.



```python
# import libraries
from scipy import stats
import matplotlib.pyplot as plt
import numpy as np

# define the rate parameter (inverse of the scale parameter)
lambda_1 = 0.5

# generate an array of x values
x = np.linspace( start: 0,  stop: 10,  num: 100)

# compute Expectation
# compute Variance
# Generate 1000 random variables
random_vars = np.random.exponential(scale=1/lambda_1, size=1000)
print(f'Expectation = {np.mean(random_vars)}')
print(f'Variance = {np.var(random_vars)}')
print(f'Random variable\n{random_vars}')

# plot the Probability Density Function of the exponential distribution
plt.plot( *args: x, stats.expon.pdf(x, scale=1/lambda_1))
plt.xlabel('Stats')
plt.ylabel('Probability')
plt.title('Exponential' + '\n' + 'PDF')
plt.show()

# plot the Cumulative Distribution Function of the exponential distribution
plt.plot( *args: x, stats.expon.cdf(x, scale=1/lambda_1))
plt.xlabel('Stats')
plt.ylabel('Probability')
plt.title('Exponential' + '\n' + 'CDF')
plt.show()

# show histogram
plt.title('Histogram' + '\n' + 'Random variable 1000 samples')
plt.xlabel('Random variable')
plt.ylabel('Frequency')
plt.hist(random_vars, bins='auto')
plt.show()
```

➢ Libraries:

`scipy.stats`: This library is utilized for statistical functions, including the probability density function (PDF) and cumulative distribution function (CDF) of the exponential distribution.

`matplotlib.pyplot`: It's essential for creating plots, enabling visualization of the probability density function, cumulative distribution function, and histograms.

`numpy`: This library is crucial for numerical operations, array creation, and generating random variables.

➢ Rate Parameter:

The rate parameter lambda_1 is introduced as the inverse of the scale parameter. It determines the rate at which events occur in the exponential distribution, influencing the shape of the distribution.

Code Execution and Statistical Analysis:

➢ Generating Random Variables:

Explain the use of numpy.random.exponential to generate 1000 random variables. The scale parameter is set as 1/lambda_1, ensuring consistency with the rate parameter. These variables simulate the time until events occur, following the exponential distribution.

➢ Expectation and Variance:

Detail the computation of expectation (mean) and variance using numpy.mean and numpy.var on the generated random variables. Emphasize that the expectation represents the average time until an event occurs, while the variance measures the spread or variability of these times.

➢ Probability Density Function (PDF) Plot:

stats.expon.pdf(x, scale=1/lambda_1): Utilizes SciPy Stats to calculate the PDF values for each x point based on the exponential distribution with the specified scale parameter.

plt.plot(x, ...): Plots these PDF values against the x values using Matplotlib.

Interpretation of the PDF: Emphasize that the PDF represents the probability density at different time values. In the exponential distribution, the PDF decreases exponentially as time increases, indicating a higher probability of shorter event times.

Decay Nature: Discuss the decay nature of the PDF, emphasizing how it reflects the decreasing likelihood of events occurring as time progresses.

- Cumulative Distribution Function (CDF) Plot:

stats.expon.cdf(x, scale=1/lambda_1): Uses SciPy Stats to compute the CDF values for each x point based on the exponential distribution with the specified scale parameter.

plt.plot(x, ...): Plots these CDF values against the x values using Matplotlib.

Understanding the CDF: Clarify that the CDF illustrates the cumulative probability of an event occurring before or at a given time. It starts from 0 and approaches 1 as time increases.

S-Shaped Curve: Discuss the S-shaped curve of the exponential CDF, indicating a gradual increase in the cumulative probability, particularly at shorter time values.

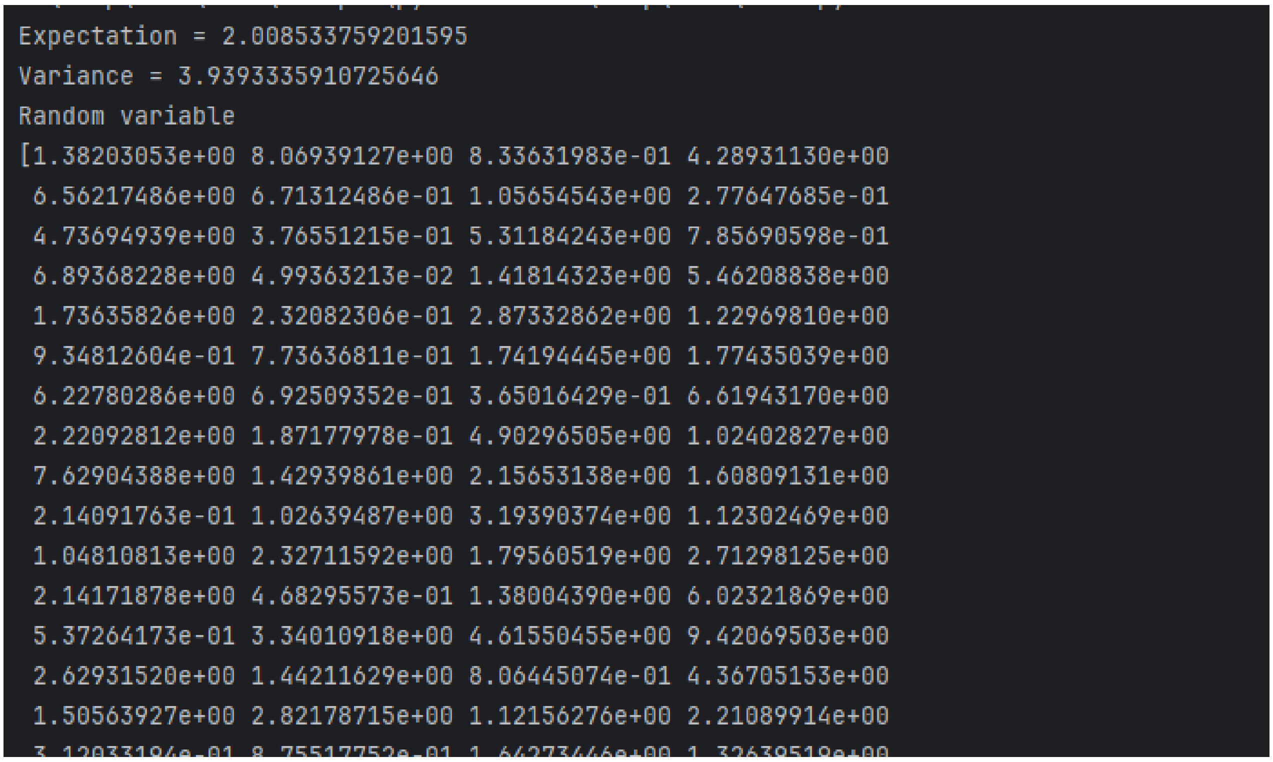
- Histogram:

plt.hist(random_vars, bins='auto'): Utilizes Matplotlib to create a histogram of the generated random variables (`random_vars`) with automatically determined bins.

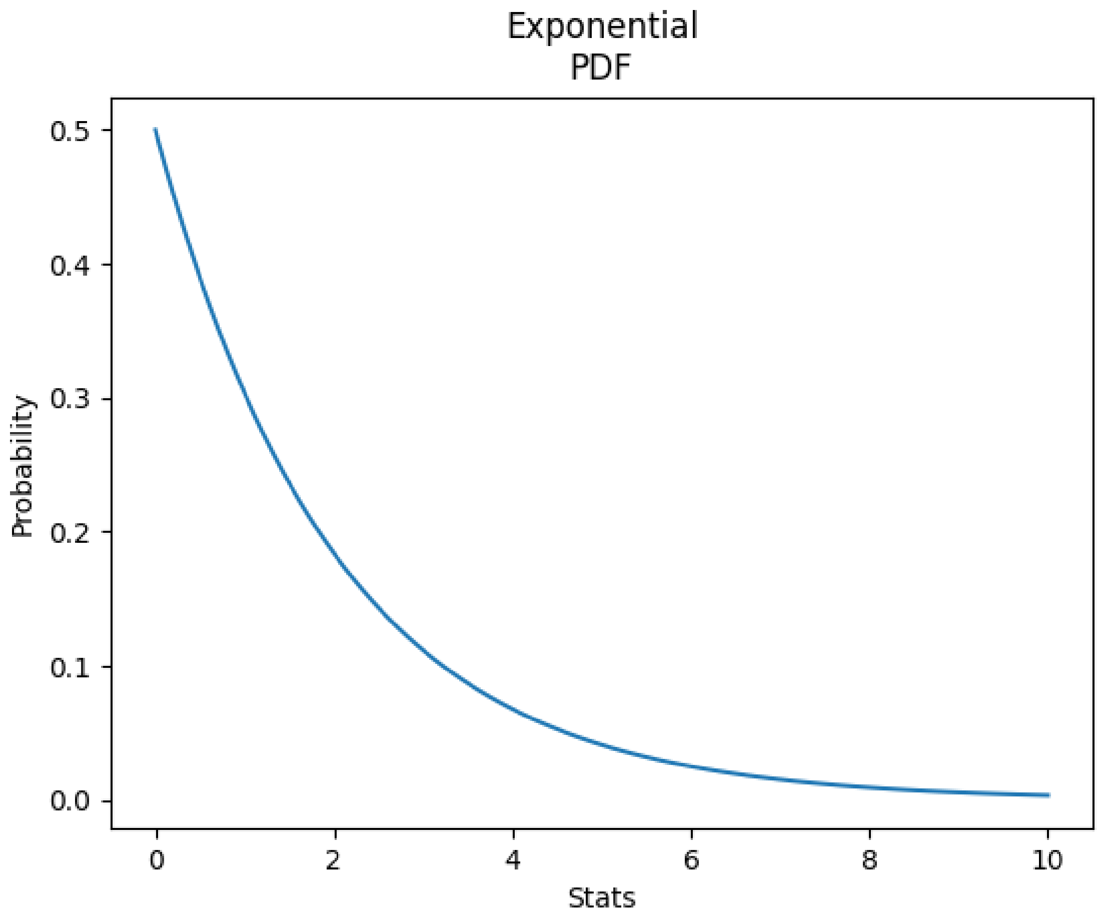
Interpreting the Histogram: Discuss that the histogram visually represents the distribution of event times. Shorter bars indicate higher frequencies of shorter event times, aligning with the exponential distribution's characteristics.

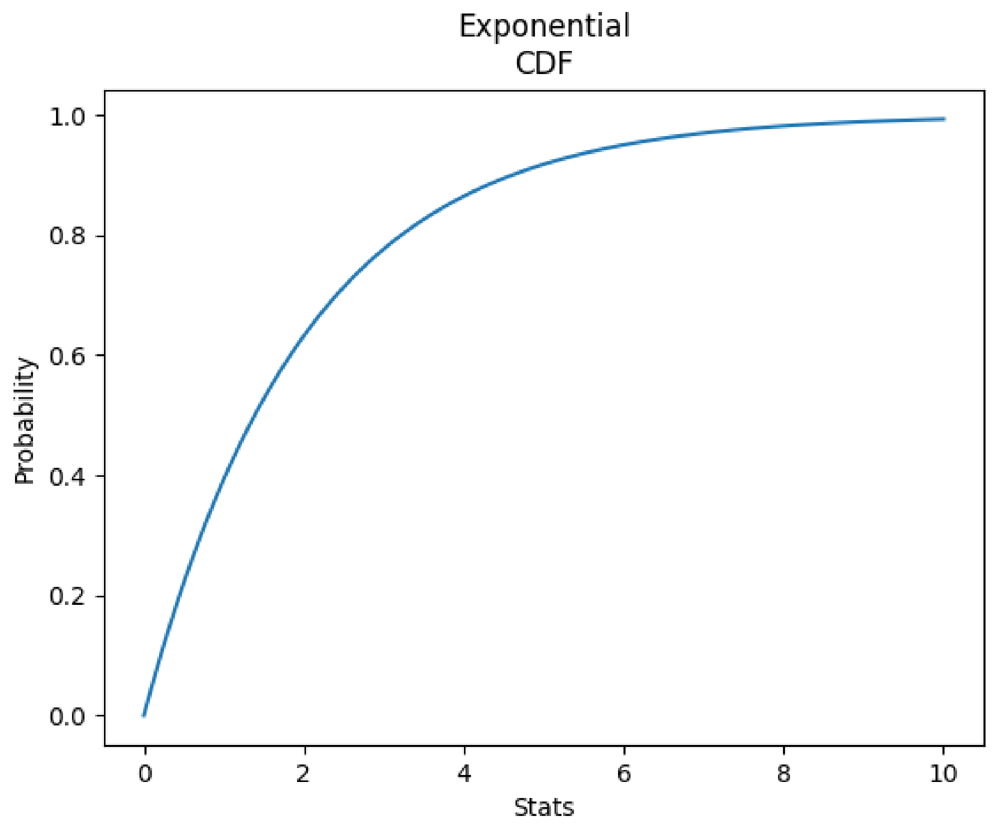
Central Limit Theorem Impact: Explain how the shape of the histogram approximates the exponential distribution, especially with a sufficiently large sample size, emphasizing the Central Limit Theorem.

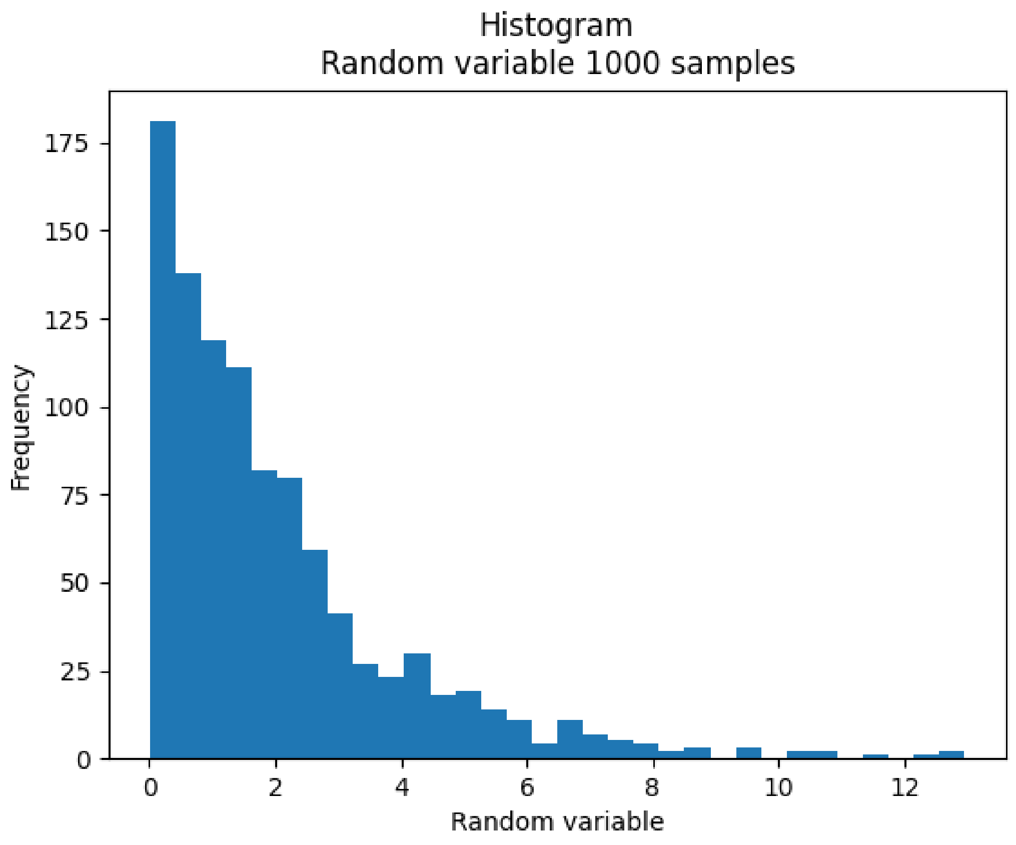
Run of code:



```
Expectation = 2.008533759201595
Variance = 3.9393335910725646
Random variable
[1.38203053e+00 8.06939127e+00 8.33631983e-01 4.28931130e+00
 6.56217486e+00 6.71312486e-01 1.05654543e+00 2.77647685e-01
 4.73694939e+00 3.76551215e-01 5.31184243e+00 7.85690598e-01
 6.89368228e+00 4.99363213e-02 1.41814323e+00 5.46208838e+00
 1.73635826e+00 2.32082306e-01 2.87332862e+00 1.22969810e+00
 9.34812604e-01 7.73636811e-01 1.74194445e+00 1.77435039e+00
 6.22780286e+00 6.92509352e-01 3.65016429e-01 6.61943170e+00
 2.22092812e+00 1.87177978e-01 4.90296505e+00 1.02402827e+00
 7.62904388e+00 1.42939861e+00 2.15653138e+00 1.60809131e+00
 2.14091763e-01 1.02639487e+00 3.19390374e+00 1.12302469e+00
 1.04810813e+00 2.32711592e+00 1.79560519e+00 2.71298125e+00
 2.14171878e+00 4.68295573e-01 1.38004390e+00 6.02321869e+00
 5.37264173e-01 3.34010918e+00 4.61550455e+00 9.42069503e+00
 2.62931520e+00 1.44211629e+00 8.06445074e-01 4.36705153e+00
 1.50563927e+00 2.82178715e+00 1.12156276e+00 2.21089914e+00
 3.12033194e-01 8.75517752e-01 1.64273446e+00 1.32638519e+00
```

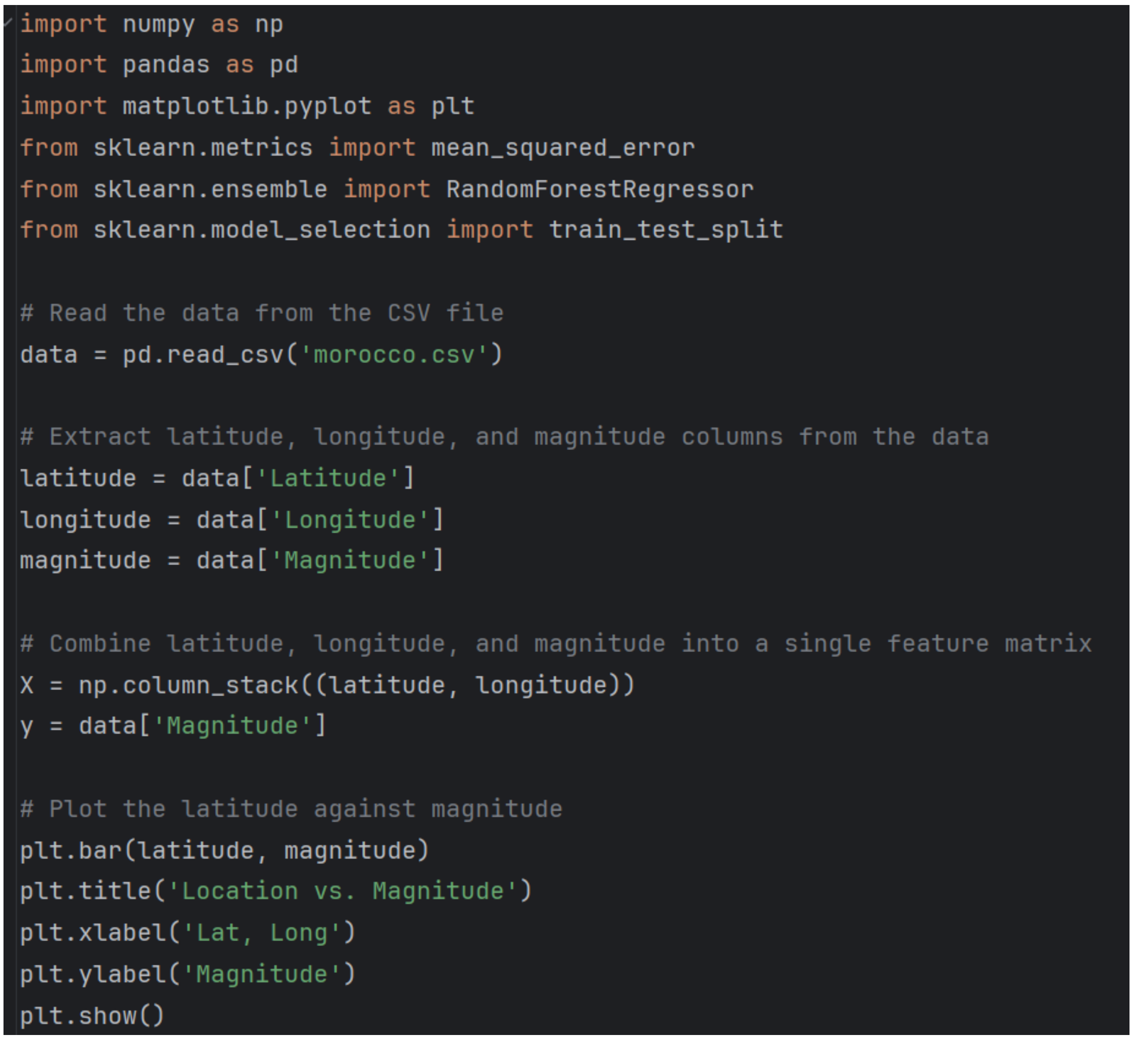



Exponential
CDF
Probability
Stats
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10

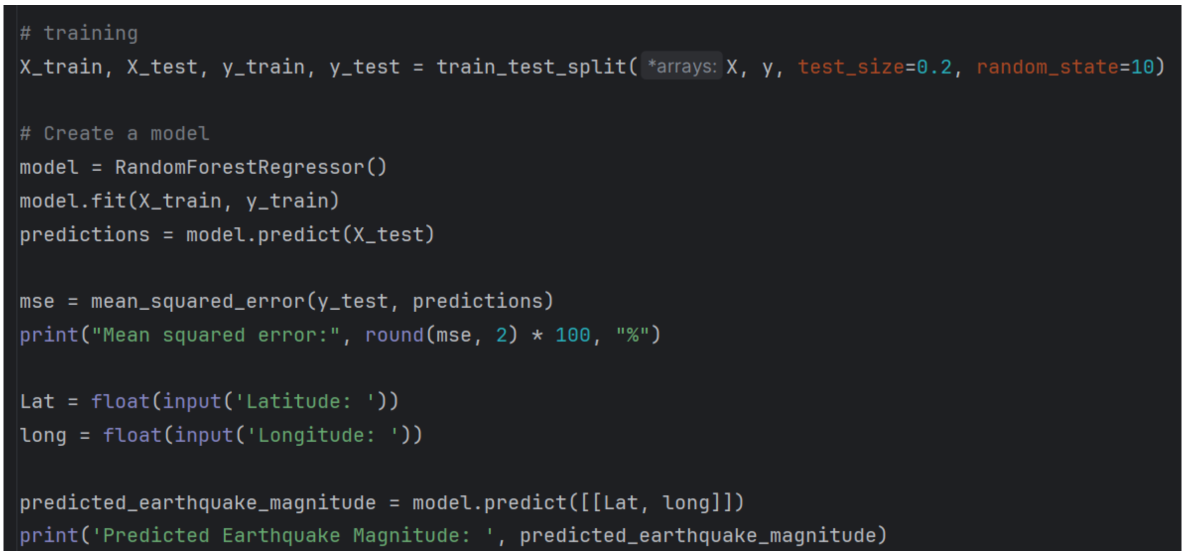

Histogram
Random variable 1000 samples
Frequency
Random variable
175
150
125
100
75
50
25
0
0
2
4
6
8
10
12

Lets begin with our first distribution example

The example is about the prediction of earthquakes in Morocco



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Read the data from the CSV file
data = pd.read_csv('morocco.csv')

# Extract latitude, longitude, and magnitude columns from the data
latitude = data['Latitude']
longitude = data['Longitude']
magnitude = data['Magnitude']

# Combine latitude, longitude, and magnitude into a single feature matrix
X = np.column_stack((latitude, longitude))
y = data['Magnitude']

# Plot the latitude against magnitude
plt.bar(latitude, magnitude)
plt.title('Location vs. Magnitude')
plt.xlabel('Lat, Long')
plt.ylabel('Magnitude')
plt.show()
```



```python
# training
X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=10)

# Create a model
model = RandomForestRegressor()
model.fit(X_train, y_train)
predictions = model.predict(X_test)

mse = mean_squared_error(y_test, predictions)
print("Mean squared error:", round(mse, 2) * 100, "%")

Lat = float(input('Latitude: '))
long = float(input('Longitude: '))

predicted_earthquake_magnitude = model.predict([[Lat, long]])
print('Predicted Earthquake Magnitude: ', predicted_earthquake_magnitude)
```

➢ **We start the code by importing some important libraries**

1. **numpy (imported as np):** NumPy is a powerful library for numerical computing .

2. **pandas (imported as pd):** Pandas is a popular library for data manipulation and analysis.

3. **matplotlib.pyplot (imported as plt)**: Matplotlib is a widely used plotting library in Python.

4. **sklearn.metrics.mean_squared_error:** This module provides the mean squared error (MSE) metric, which is commonly used in regression tasks to evaluate the performance of regression models. The mean squared error measures the average squared difference between the predicted and true values of a target variable. Lower values of MSE indicate better model performance.

5. **sklearn.ensemble.RandomForestRegressor**: This module provides the Random Forest Regressor algorithm, which is an ensemble learning method for regression tasks and features. A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The distRforest package is based on the rpart package and extends its functionality. The rpart package implements classification trees, standard regression trees (with a squared error loss function), Poisson regression trees (with the Poisson deviance as a loss function) and exponential scaling for survival data. This package is mainly intented for users who want to develop a **random forest** on **count data** (via the Poisson distribution) or on **long-tailed data** (via the gamma or log-normal distribution)

6. **sklearn.model_selection.train_test_split**: This module provides a function to split a dataset into training and testing subsets. It is commonly used to evaluate the performance of machine learning models; you can train the model on the training set and evaluate its performance on the unseen testing set. This helps to estimate how well the model generalizes to new, unseen data.

➢ **Read data from csv file**
We use pd.read_csv to read content of csv file, pandas library provide this function to read csv file and create dataframe. Then we store the content of file in data variable

➢ **Extract the Latitude, Longitude, and magnitude columns from the data**
This code assumes that the data DataFrame has columns named "Latitude", "Longitude", and "Magnitude". By accessing these columns using square brackets, you can assign them to separate variables (latitude, longitude, and magnitude).

After executing this code, you will have three separate Series objects (latitude, longitude, and magnitude) that contain the values from their respective columns in the data DataFrame. You can then perform further operations or analysis on these Series objects as needed.

- **combining the latitude and longitude columns into a single feature matrix**.

we use the column_stack() function from the NumPy library (np) to stack the latitude and longitude arrays (or Series) together horizontally, creating a 2D array

then assign the values from the “magnitude” column of the data DataFrame to the variable y.

By combining the latitude and longitude into a single feature matrix (X) and keeping the magnitude values in a separate variable (y), you are preparing the data for a machine learning task, such as regression or clustering, where the goal is to predict the magnitude based on the latitude and longitude information

- **plotting latitude against magnitude**

use plt.bar(latitude, magnitude) to create a bar plot using the bar() function from pyplot. The latitude values are used as the x-axis values, and the magnitude values are used as the corresponding bar heights.

Use plt.title('Location vs. Magnitude'): This line sets the title of the plot to "Location vs. Magnitude" using the title() function.

use plt.xlabel('Latitude') to set the label for the x-axis and plt.ylabel('Magnitude') to set the label for the y-axis and plt.show() to display the plot

- **training**

here we split the data into training and testing sets using the train_test_split function .This function is part of the sklearn.model_selection module and is used to split data into training and testing subsets. It randomly shuffles the data and divides it into two sets based on the specified test_size parameter.

The *arrays syntax is used to pass multiple arrays or datasets to the train_test_split function so X, y arrays are passed.

test_size=0.2: This parameter specifies the proportion of the data that will be allocated to the testing set. In this case, 20% of the data will be assigned to the testing set, while the remaining 80% will be used for training.

random_state=10: This parameter sets the random seed value to ensure reproducibility. By specifying a fixed random_state, the data split will be the same every time the code is executed.

- **Creating a model**

we use RandomForestRegressor() that is a machine learning model from the scikit-learn library that is based on the random forest algorithm and is used for regression tasks, then assign it to variable called model

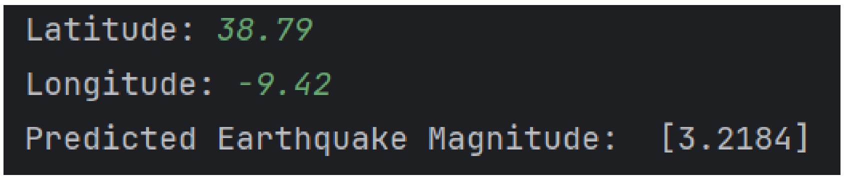
Then we train the model using the fit() method that takes the training data X_train (features) and y_train (target variable) as input and fits the model to the training data.

Then we generate predictions using the trained model on the testing data X_test. The predict() method takes the testing data as input and returns the predicted values for the target variable. And assign it to variable called predictions .

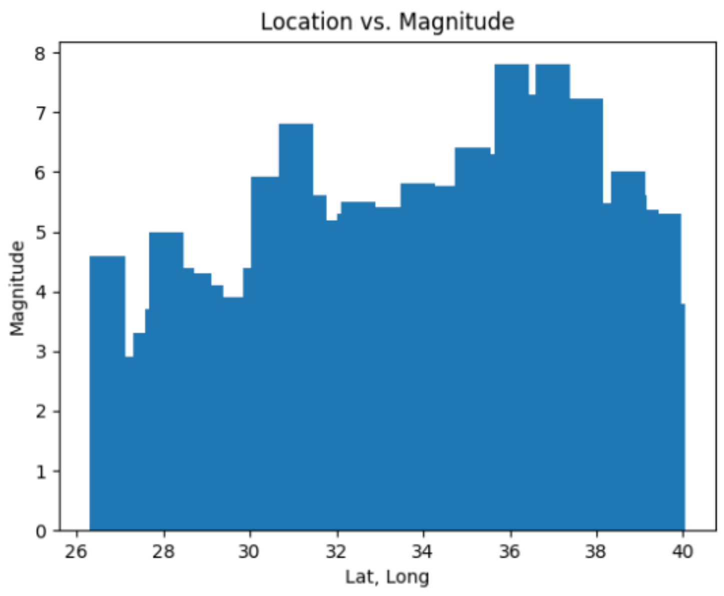
To calculate the error percentage use mean_squared_error that takes y_tests and prediction to estimate the error percentage then print it

Overall, here we use this code to input latitude and longitude and calculate the predicted earthquake magnitude using function model.predict

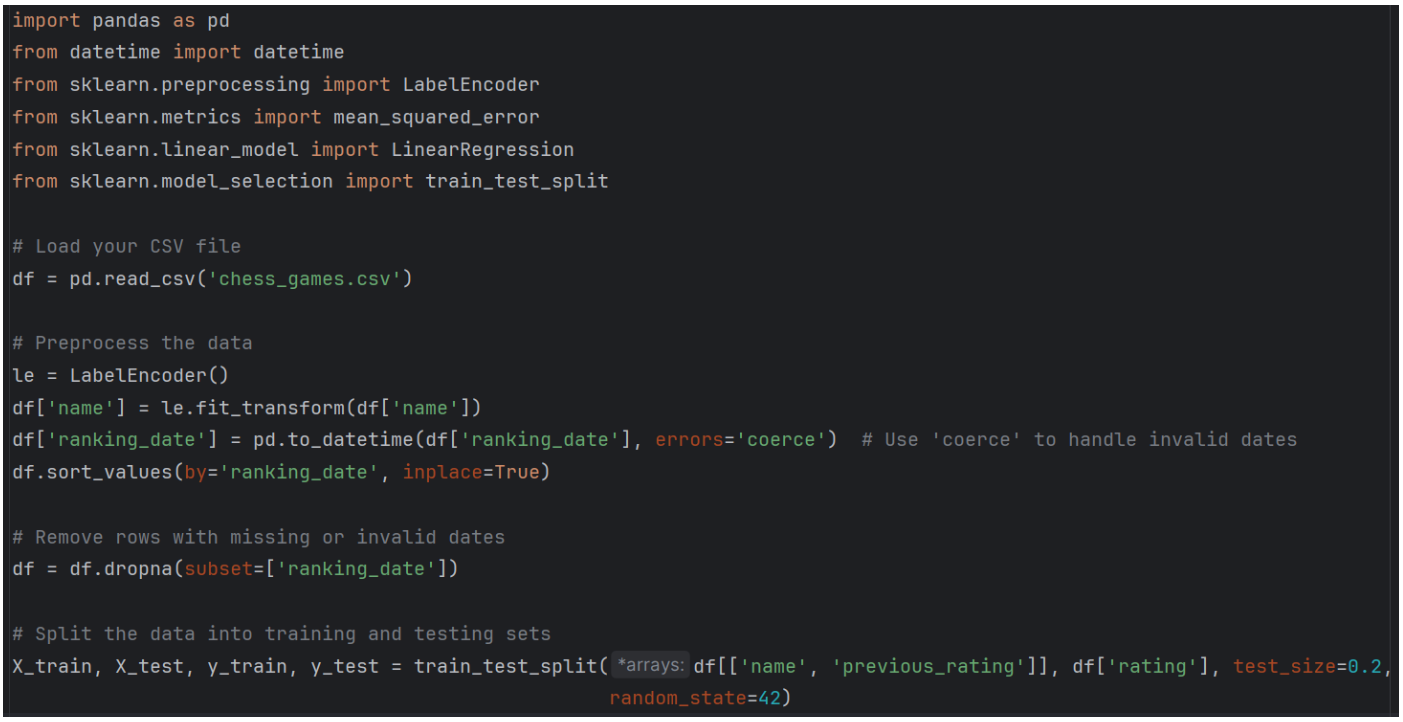
Let's see a sample of it



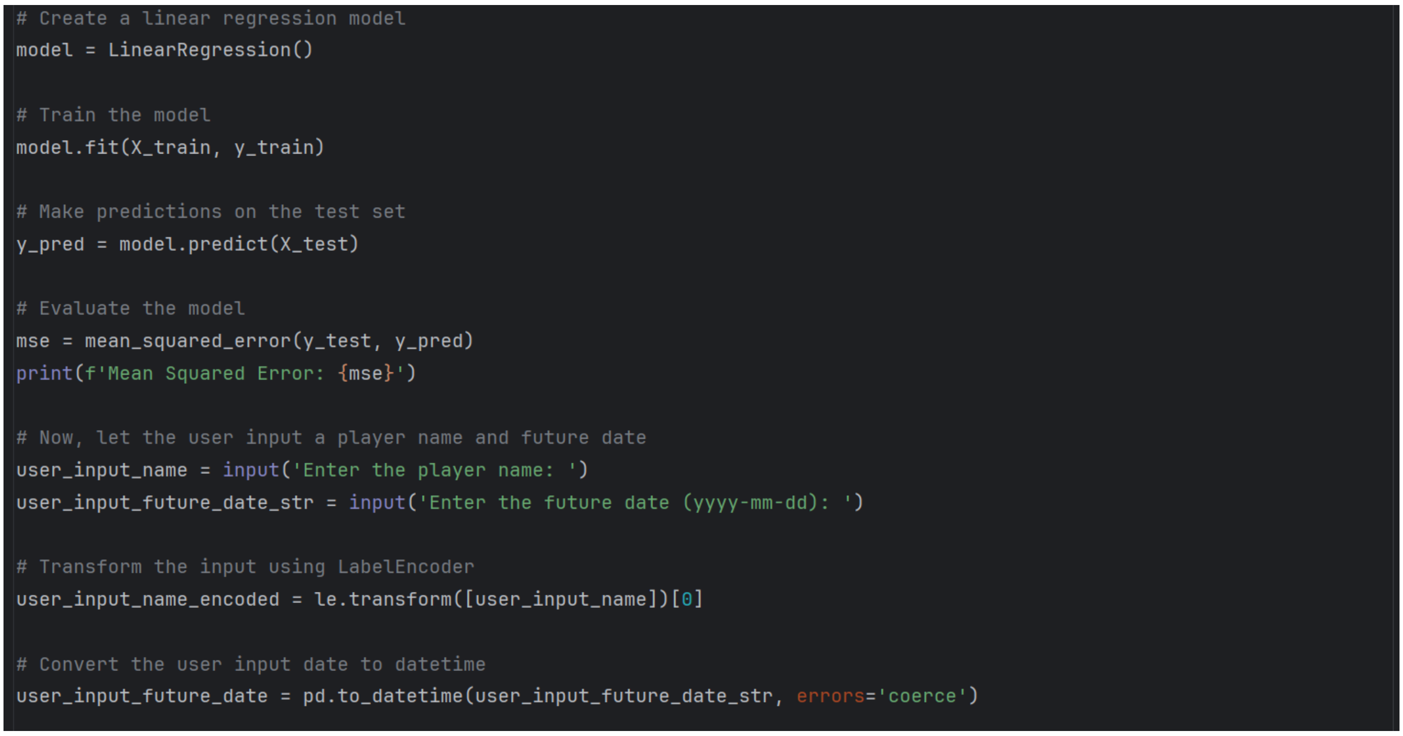
```
Latitude: 38.79
Longitude: -9.42
Predicted Earthquake Magnitude:  [3.2184]
```



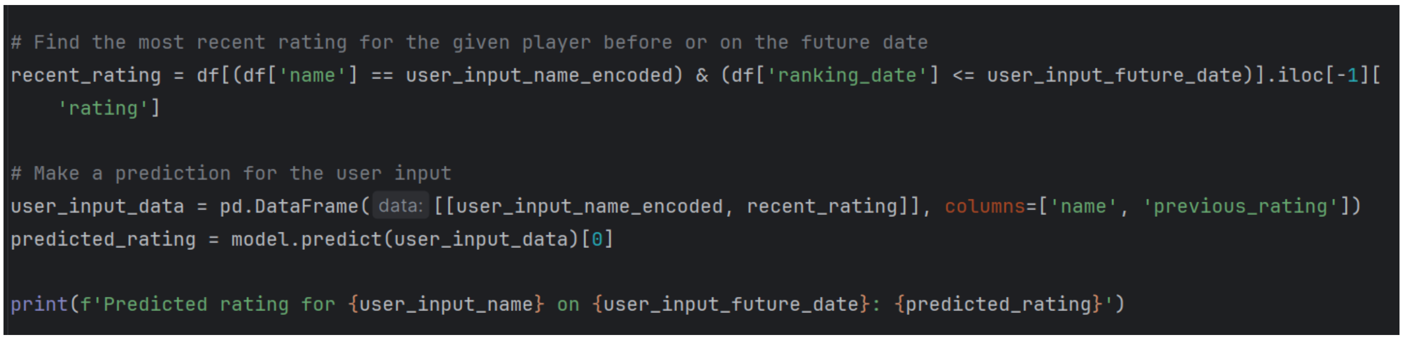
The second example is talking about prediction of chess players future rate



```python
import pandas as pd
from datetime import datetime
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Load your CSV file
df = pd.read_csv('chess_games.csv')

# Preprocess the data
le = LabelEncoder()
df['name'] = le.fit_transform(df['name'])
df['ranking_date'] = pd.to_datetime(df['ranking_date'], errors='coerce')  # Use 'coerce' to handle invalid dates
df.sort_values(by='ranking_date', inplace=True)

# Remove rows with missing or invalid dates
df = df.dropna(subset=['ranking_date'])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split( *arrays: df[['name', 'previous_rating']], df['rating'], test_size=0.2,
                                                    random_state=42)
```



```python
# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Now, let the user input a player name and future date
user_input_name = input('Enter the player name: ')
user_input_future_date_str = input('Enter the future date (yyyy-mm-dd): ')

# Transform the input using LabelEncoder
user_input_name_encoded = le.transform([user_input_name])[0]

# Convert the user input date to datetime
user_input_future_date = pd.to_datetime(user_input_future_date_str, errors='coerce')
```


```
# Find the most recent rating for the given player before or on the future date
recent_rating = df[(df['name'] == user_input_name_encoded) & (df['ranking_date'] <= user_input_future_date)].iloc[-1][
    'rating']

# Make a prediction for the user input
user_input_data = pd.DataFrame( data: [[user_input_name_encoded, recent_rating]], columns=['name', 'previous_rating'])
predicted_rating = model.predict(user_input_data)[0]

print(f'Predicted rating for {user_input_name} on {user_input_future_date}: {predicted_rating}')
```

- **Import Libraries:**

pandas: Used for handling the dataset as a DataFrame, making it easier to manipulate and analyze tabular data.

datetime: Enables working with date and time data, crucial for handling the 'ranking_date' column.

LabelEncoder from sklearn.preprocessing: Converts categorical variables like player names into numerical format for machine learning models.

mean_squared_error from sklearn.metrics: Measures the average squared difference between actual and predicted values, providing a quantitative measure of model performance.

LinearRegression from sklearn.linear_model: Implements the linear regression model, a simple method for predicting a continuous outcome based on one or more predictor variables.

train_test_split from sklearn.model_selection: Splits the dataset into training and testing sets for model evaluation.

- **Load and Preprocess Data:**

pd.read_csv('chess_games.csv'): Reads the CSV file into a pandas DataFrame named df.

LabelEncoder() and le.fit_transform(): Encodes the 'name' column, converting player names into numerical labels.

pd.to_datetime(df['ranking_date'], errors='coerce'): Converts the 'ranking_date' column to datetime format, handling invalid dates with 'coerce'.

df.sort_values(by='ranking_date', inplace=True): Sorts the DataFrame by 'ranking_date'.

df = df.dropna(subset=['ranking_date']): Removes rows with missing or invalid dates.

- **Split Data and Create Model:**

train_test_split(): Divides the dataset into training and testing sets (`X_train`, X_test, y_train, `y_test`) for model evaluation.

LinearRegression(): Initializes a linear regression model named model.

- **Train and Evaluate Model:**

model.fit(X_train, y_train): Trains the linear regression model using the training data.

model.predict(X_test): Generates predictions on the test set.

mean_squared_error(y_test, y_pred): Calculates the Mean Squared Error between the actual and predicted ratings, evaluating the model's accuracy.

- **User Input and Prediction:**

input(): Accepts user input for player name and a future date in 'yyyy-mm-dd' format.

le.transform([user_input_name])[0]: Converts the user-input player name into the corresponding numerical label.

pd.to_datetime(user_input_future_date_str, errors='coerce'): Converts the user-input future date to datetime format.

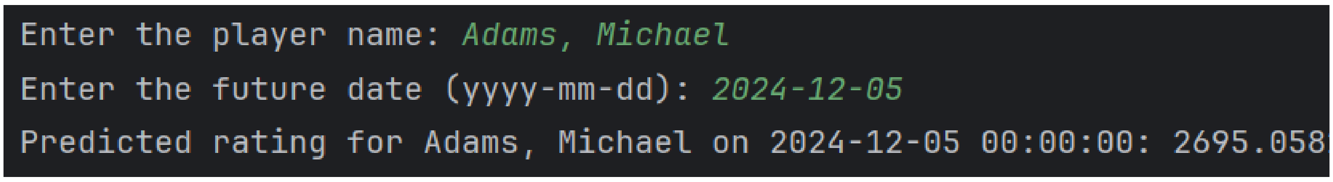
df[(df['name'] == user_input_name_encoded) & (df['ranking_date'] <= user_input_future_date)].iloc[-1]['rating']: Retrieves the most recent rating for the specified player before or on the future date.

pd.DataFrame([[user_input_name_encoded, recent_rating]], columns=['name', 'previous_rating']): Creates a DataFrame with the encoded player name and recent rating for prediction.

model.predict(user_input_data)[0]: Uses the trained model to predict the player's rating on the specified future date.

- **Output:**

print(f'Predicted rating for {user_input_name} on {user_input_future_date}: {predicted_rating}'): Displays the predicted rating for the user-input player on the user-input future date.



```
Enter the player name: Adams, Michael
Enter the future date (yyyy-mm-dd): 2024-12-05
Predicted rating for Adams, Michael on 2024-12-05 00:00:00: 2695.058
```

This script essentially follows the typical machine learning workflow: data preprocessing, model training, evaluation, and making predictions based on user input. It utilizes a linear regression model to predict chess player ratings.

References

Books:

- Chan, Stanley H. Introduction to Probability for Data Science. Michigan Publishing Services, 2021.

Websites:

- https://henckr.github.io/distRforest/
- https://www.kaggle.com/datasets/odartey/top-chess-players
- https://www.kaggle.com/datasets/usgs/earthquake-database
- Zach. "5 Real-Life Examples of the Binomial Distribution." Statology, 14 July 2021, https://www.statology.org/binomial-distribution-real-life-examples/.
- "Mean and Variance of Binomial Distribution | Formulas, Definition, Examples." Toppr Answr, https://www.toppr.com/ask/en-us/content/concept/mean-and-variance-of-binomial-distribution-207672/.
- https://stats.stackexchange.com/questions/148803/how-does-linear-regression-use-the-normal-distribution

<u>**Task assignment:**</u>

The tasks were divided into four sections that are:

1. Searching for the content.

2. Coding & Testing.

3. Typing the report.

4. Designing Presentation.

Team members' percentage for each task:

Searching for the content.

Abdelwahab Alaa	20%
Rana Ehab	20%
Shahd Amgad	20%
Omar Mohamed Adel	20%
Sherif Mostafa	20%

Coding & Testing.

Abdelwahab Alaa	30%
Rana Ehab	10%
Shahd Amgad	10%
Omar Mohamed Adel	20%
Sherif Mostafa	30%

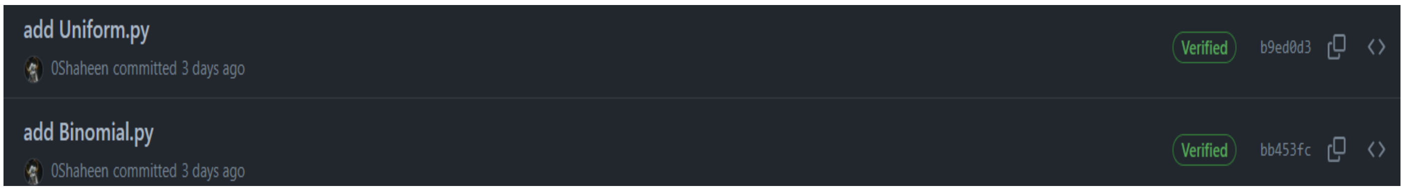
Typing the report.

Abdelwahab Alaa	20%
Rana Ehab	30%
Shahd Amgad	30%
Omar Mohamed Adel	10%
Sherif Mostafa	10%

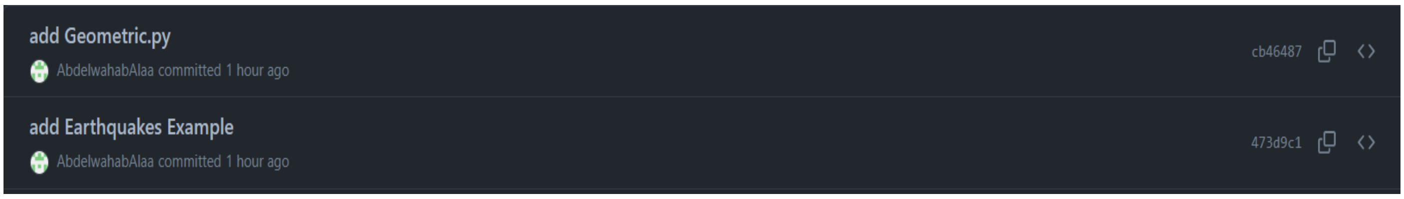
Designing Presentation.

Abdelwahab Alaa	25%
Rana Ehab	15%
Shahd Amgad	15%
Omar Mohamed Adel	30%
Sherif Mostafa	15%

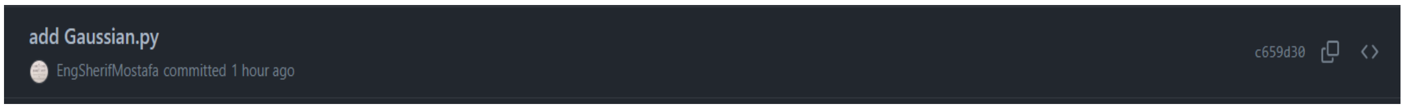
Omar Shaheen’s Contribution:

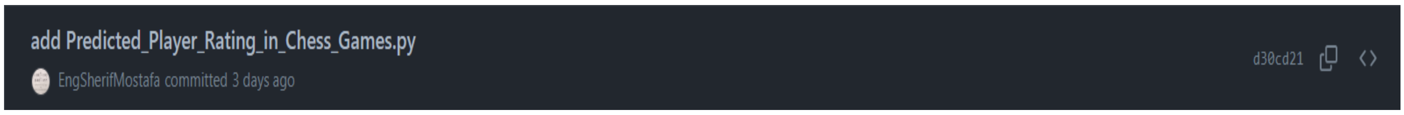

add Uniform.py
OShaheen committed 3 days ago · Verified · b9ed0d3
add Binomial.py
OShaheen committed 3 days ago · Verified · bb453fc

Abdelwahab Alaa’s Contribution:

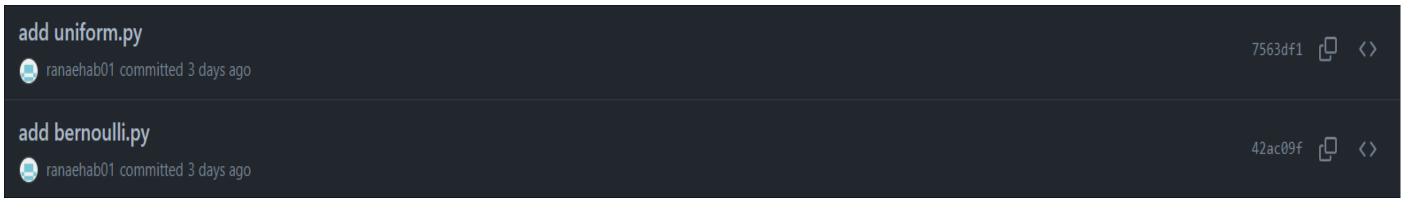

add Geometric.py
AbdelwahabAlaa committed 1 hour ago · cb46487
add Earthquakes Example
AbdelwahabAlaa committed 1 hour ago · 473d9c1

Sherif Mostafa’s Contribution:

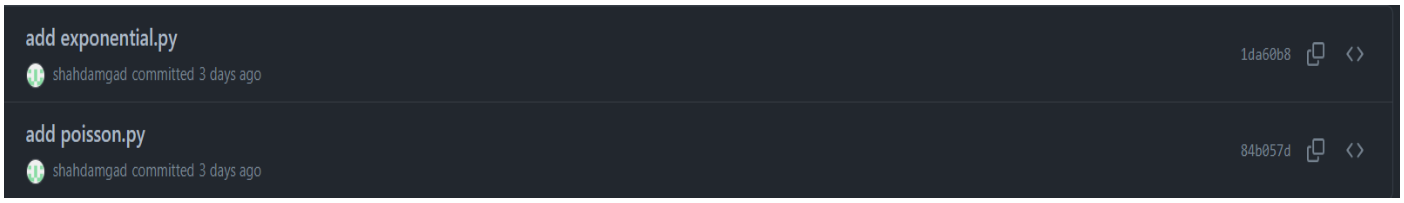

add Gaussian.py
EngSherifMostafa committed 1 hour ago · c659d30


add Predicted_Player_Rating_in_Chess_Games.py
EngSherifMostafa committed 3 days ago · d30cd21

Rana Ehab’s Contribution:


add uniform.py
ranaehab01 committed 3 days ago · 7563df1
add bernoulli.py
ranaehab01 committed 3 days ago · 42ac09f

Shahd Amgad’s Contribution:


add exponential.py
shahdamgad committed 3 days ago · 1da60b8
add poisson.py
shahdamgad committed 3 days ago · 84b057d