

March 2024

# STATISTIC PROBABILITY IN PYTHON

Presented by TIMOTHEE NKWAR

INSTRUCTOR : Dr. Aysegul Erem



# 1. PYTHON



Python is a programming language that's:

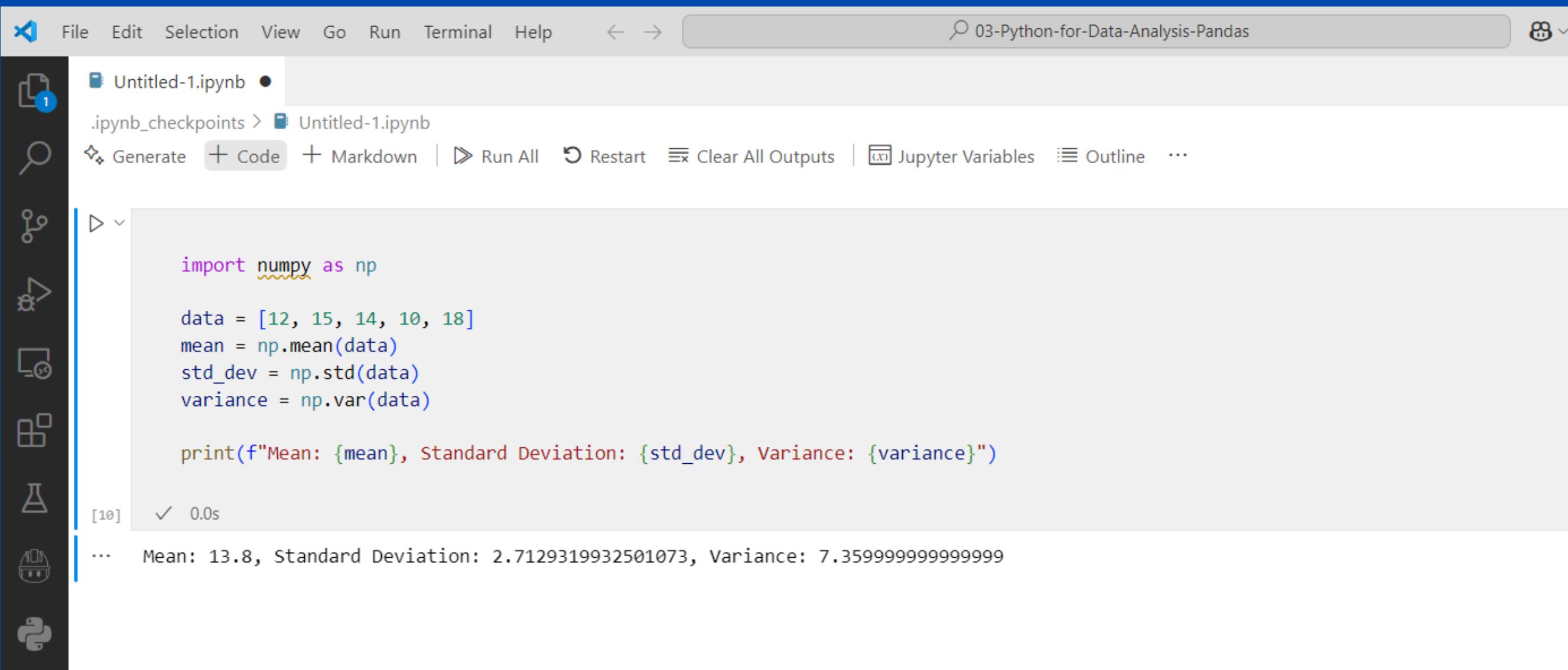
- ✓ Easy to read and learn.
- ✓ Versatile – used for Data Science, Web Development, Automation, AI, Games, and more.
- ✓ Backed by a massive community with thousands of ready-to-use libraries.

# SOME PYTHON LIBRARIES FOR STATISTICS?

Libraries	What it's for
Numpy	Mathematical & statistical operations, arrays, basic stats (mean, median, variance) .
scipy.stats	Advanced statistics, distributions, hypothesis testing.
Pandas	Data manipulation + summary statistics.
matplotlib & seaborn	Visualization of distributions, boxplots, histograms.

# a. NumPy - Core Stats

NumPy is your foundation for numerical  
and statistical operations:



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** 03-Python-for-Data-Analysis-Pandas.
- Toolbar:** .ipynb checkpoints > Untitled-1.ipynb, Generate, + Code, + Markdown, ▶ Run All, ⚡ Restart, Clear All Outputs, Jupyter Variables, Outline, ...
- Code Cell:** Contains Python code for importing numpy, calculating mean, standard deviation, and variance, and printing the results.

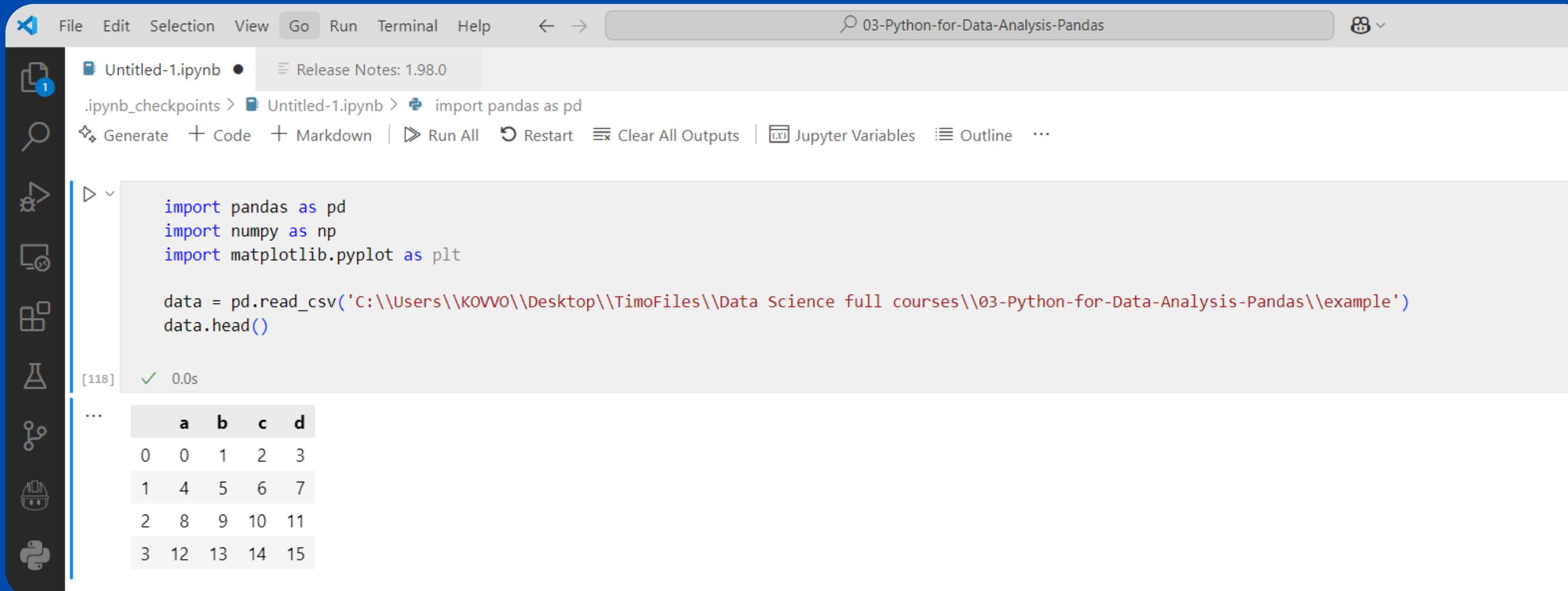
```
import numpy as np

data = [12, 15, 14, 10, 18]
mean = np.mean(data)
std_dev = np.std(data)
variance = np.var(data)

print(f"Mean: {mean}, Standard Deviation: {std_dev}, Variance: {variance}")
```
- Output Cell:** Shows the execution status [10] and the output: Mean: 13.8, Standard Deviation: 2.7129319932501073, Variance: 7.359999999999999.
- Sidebar:** Includes icons for file, search, refresh, and other notebook functions.

# c. pandas - Data Analysis

pandas is for working with dataframes and generating quick statistics:  
This is how to load data with pandas .



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** 03-Python-for-Data-Analysis-Pandas.
- Toolbar:** Includes icons for file operations, search, generate, code, markdown, run all, restart, clear outputs, jupyter variables, and outline.
- Code Cell:** Contains Python code:

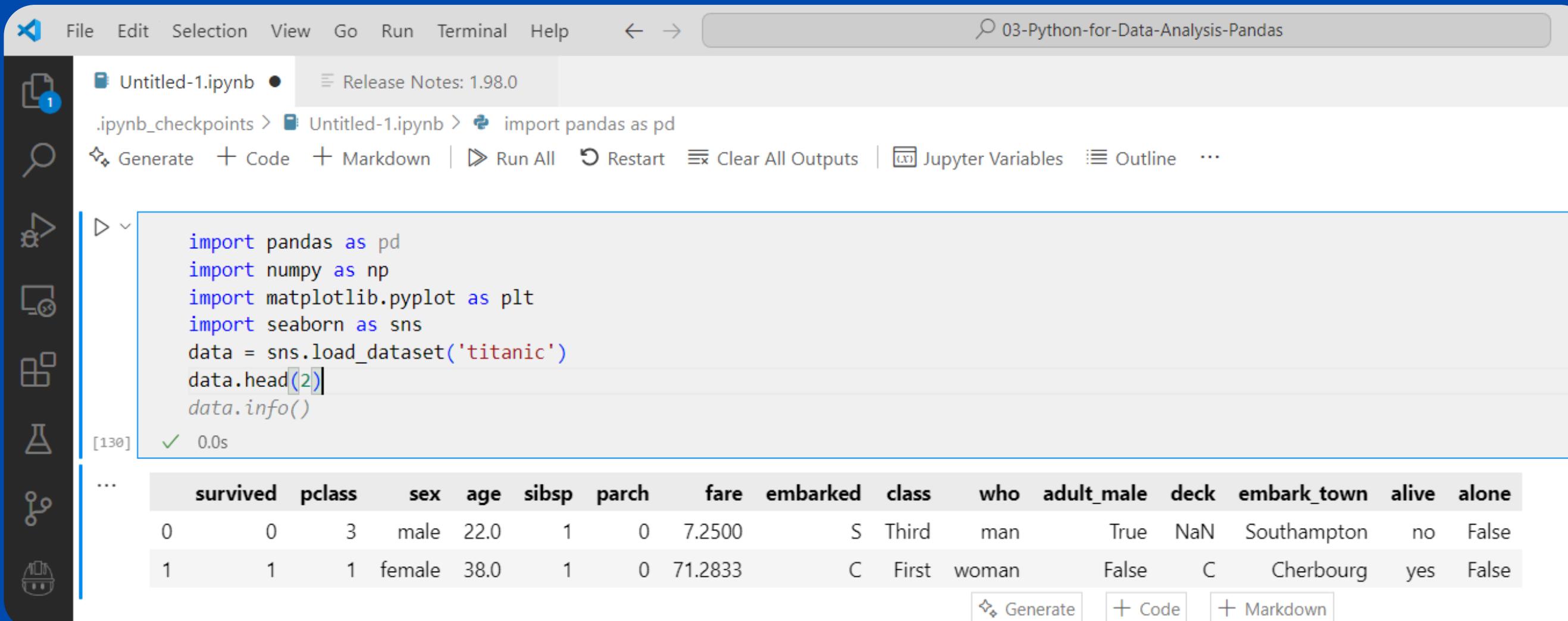
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_csv('C:\\\\Users\\\\KOVVO\\\\Desktop\\\\TimoFiles\\\\Data Science full courses\\\\03-Python-for-Data-Analysis-Pandas\\\\example')
data.head()
```
- Output Cell:** Shows the execution time [118] 0.0s and the resulting DataFrame:

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

# d. seaborn & matplotlib

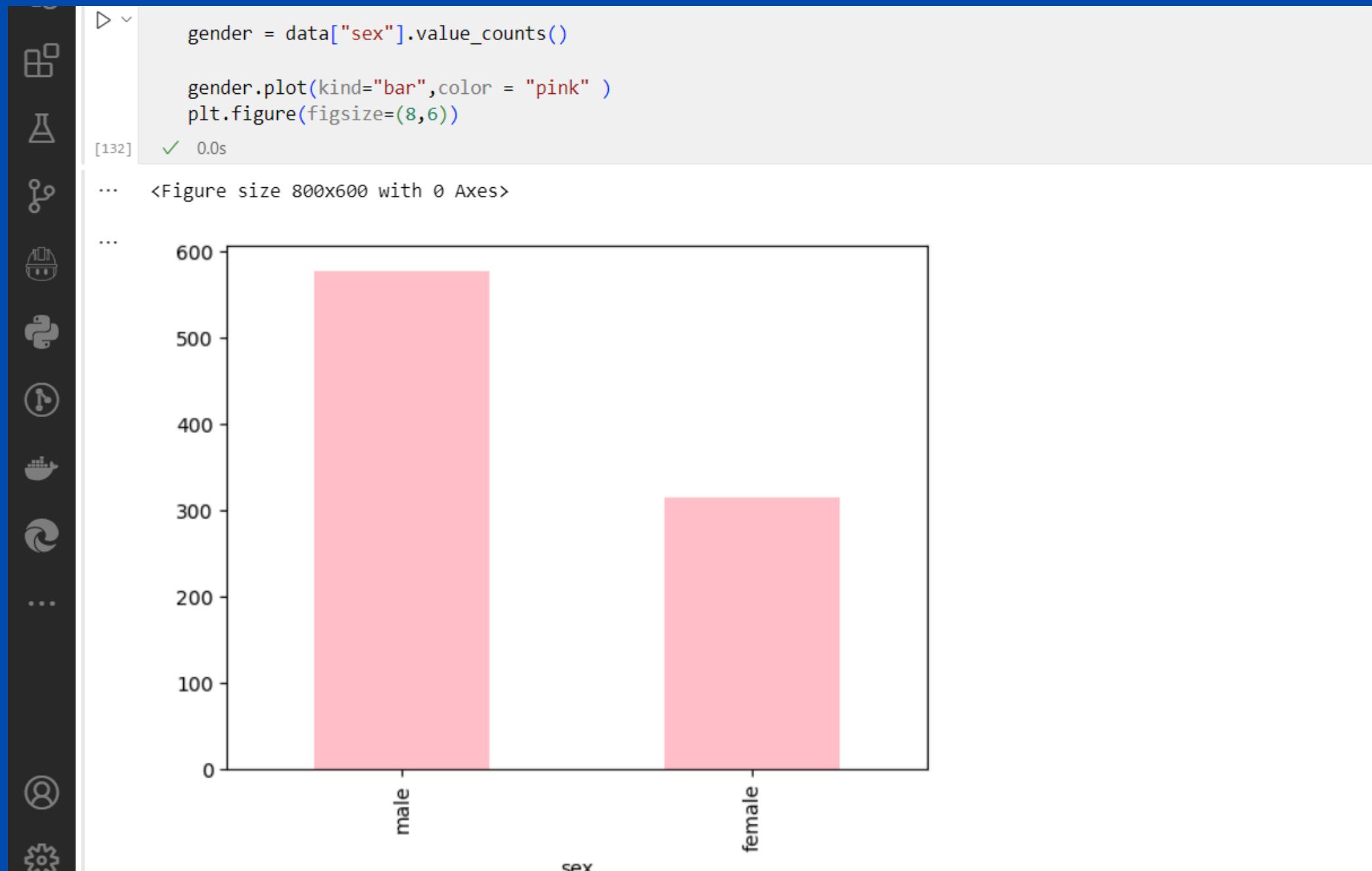
For visualizing distributions, correlations , and trends , seaborn datasets :



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** 03-Python-for-Data-Analysis-Pandas.
- Toolbar:** .ipynb checkpoints, Untitled-1.ipynb, import pandas as pd, Generate, + Code, + Markdown, Run All, Restart, Clear All Outputs, Jupyter Variables, Outline, ...
- Code Cell:** Contains Python code to import pandas, numpy, matplotlib.pyplot, and seaborn, then load the titanic dataset and print its head. The output shows the first two rows of the titanic dataset.
- Data Preview:** Shows the first two rows of the titanic dataset with columns: survived, pclass, sex, age, sibsp, parch, fare, embarked, class, who, adult\_male, deck, embark\_town, alive, alone.
- Bottom Buttons:** Generate, + Code, + Markdown.

# d.matplotlib - Visual Stats



## 2. RANDOM VARIABLE

A RANDOM VARIABLE IS A VARIABLE WHOSE VALUES ARE DETERMINED BY THE OUTCOMES OF A RANDOM PROCESS OR EXPERIMENT. IT CAN TAKE DIFFERENT VALUES BASED ON CHANCE AND IS TYPICALLY ASSOCIATED WITH A PROBABILITY DISTRIBUTION.

THERE ARE TWO TYPES OF RANDOM VARIABLES:

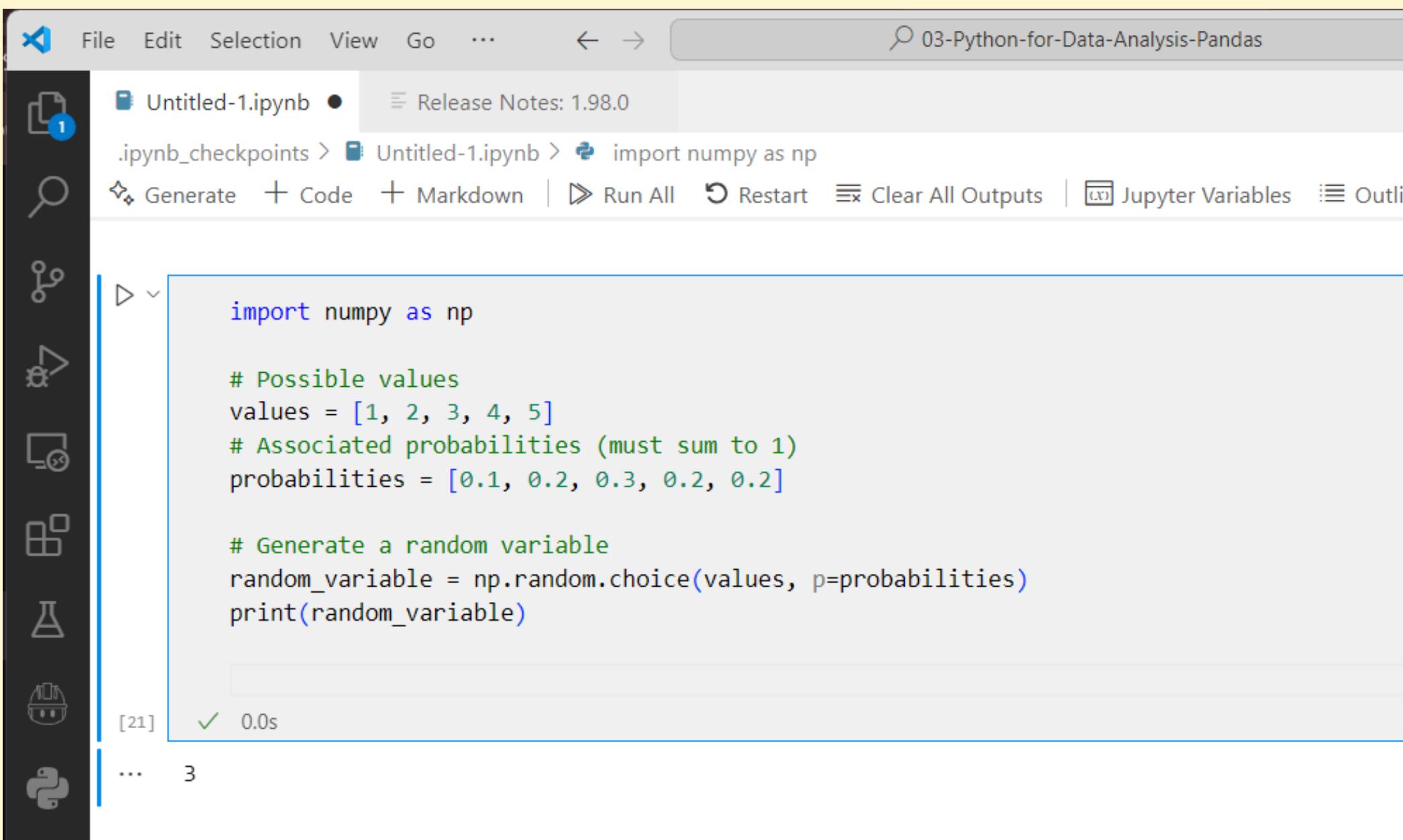
DISCRETE RANDOM VARIABLE: TAKES ON A FINITE OR COUNTABLE NUMBER OF VALUES. FOR EXAMPLE, THE NUMBER OF HEADS IN A COIN TOSS (EITHER 0 OR 1)

CONTINUOUS RANDOM VARIABLE: TAKES ON AN INFINITE NUMBER OF POSSIBLE VALUES WITHIN A GIVEN RANGE. FOR EXAMPLE, THE HEIGHT OF A PERSON, WHICH CAN BE ANY REAL NUMBER WITHIN A CERTAIN RANGE.

# DISCRETE RANDOM VARIABLE

DISCRETE RANDOM VARIABLE: TAKES ON A FINITE OR COUNTABLE NUMBER OF VALUES.  
FOR EXAMPLE, THE NUMBER OF HEADS IN A COIN TOSS (EITHER 0 OR 1) .

DICE ROLL: {1, 2, 3, 4, 5, 6}



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** 03-Python-for-Data-Analysis-Pandas
- File Menu:** File, Edit, Selection, View, Go, ...
- Toolbar:** Release Notes: 1.98.0, .ipynb\_checkpoints, Untitled-1.ipynb, import numpy as np, Generate, + Code, + Markdown, Run All, Restart, Clear All Outputs, Jupyter Variables, Outline.
- Code Cell:** The cell contains the following Python code:

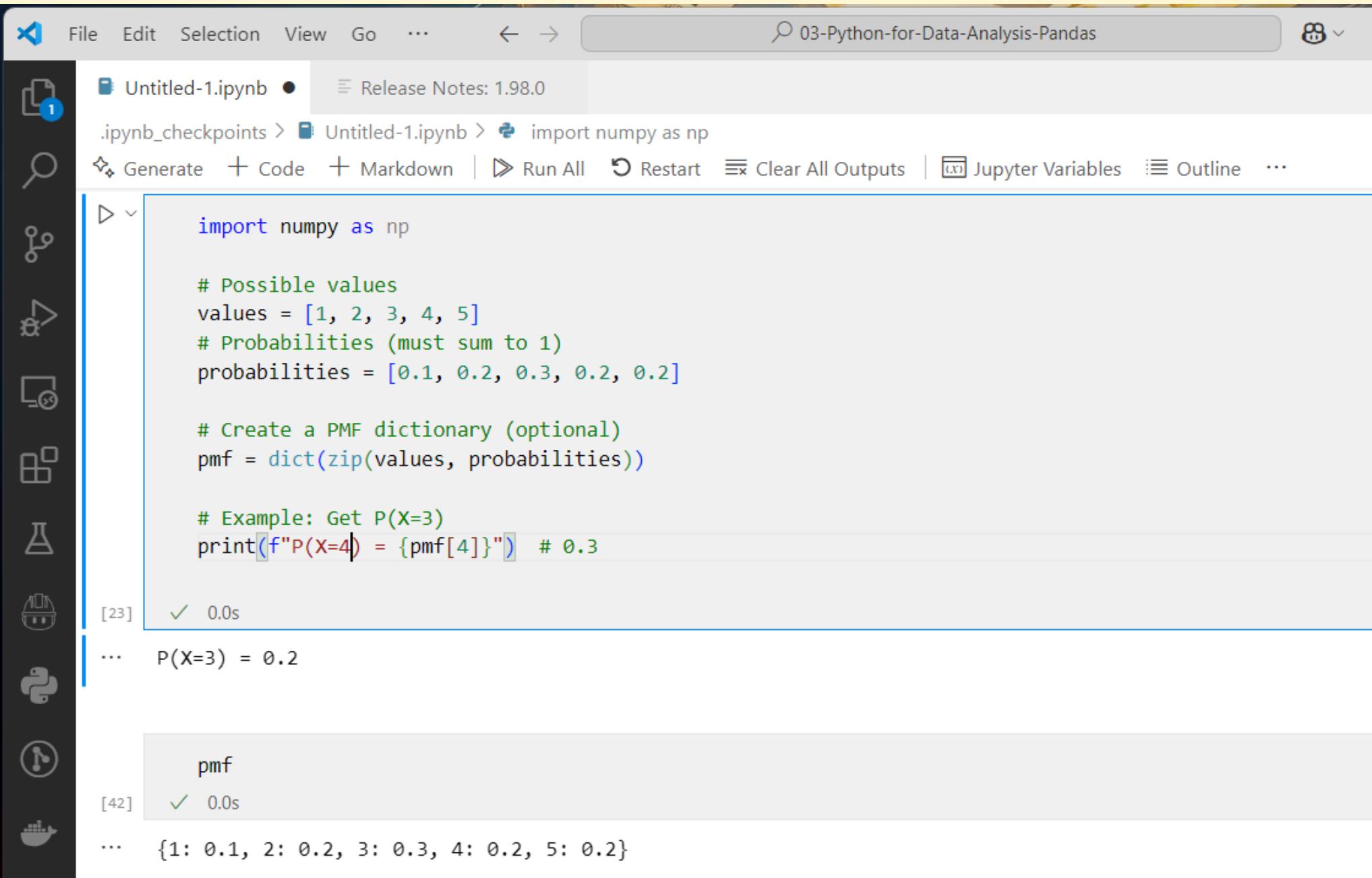
```
import numpy as np

# Possible values
values = [1, 2, 3, 4, 5]
# Associated probabilities (must sum to 1)
probabilities = [0.1, 0.2, 0.3, 0.2, 0.2]

# Generate a random variable
random_variable = np.random.choice(values, p=probabilities)
print(random_variable)
```
- Output:** The output of the cell is a single digit: 0.0s, indicating the result of the print statement.
- Cell Number:** [21]
- Page Number:** 3

# PMF OF DISCRETE RANDOM VARIABLE

if you want the probability of a specific value like the  $P(X=3)$   
Here how to do that:



The screenshot shows a Jupyter Notebook interface with the following code:

```
import numpy as np

# Possible values
values = [1, 2, 3, 4, 5]
# Probabilities (must sum to 1)
probabilities = [0.1, 0.2, 0.3, 0.2, 0.2]

# Create a PMF dictionary (optional)
pmf = dict(zip(values, probabilities))

# Example: Get P(X=3)
print(f"P(X=3) = {pmf[3]}") # 0.3
```

The output of the first cell is:

```
[23]    ✓  0.0s
...     P(X=3) = 0.2
```

The output of the second cell is:

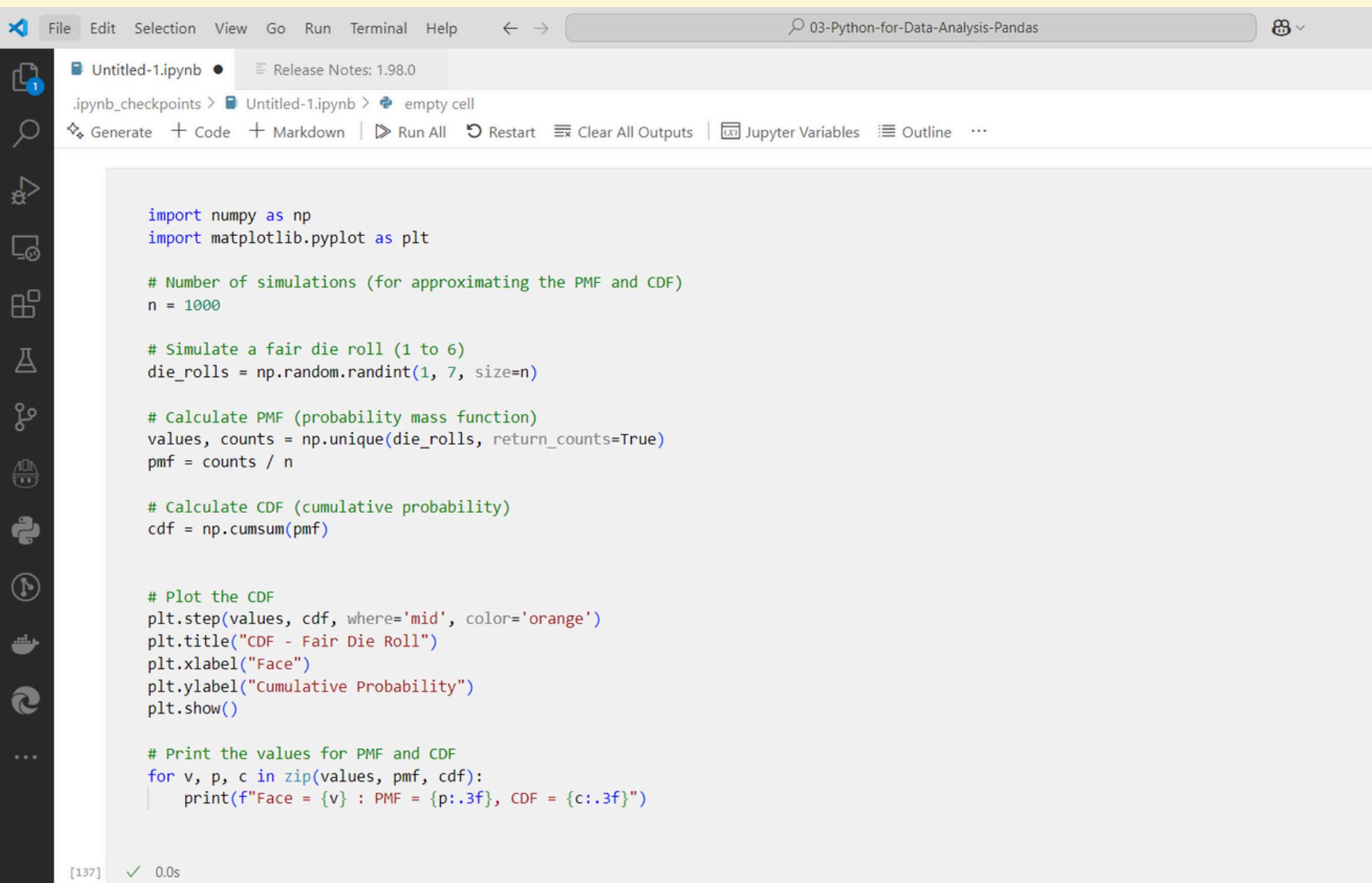
```
[42]    ✓  0.0s
...     pmf
...     {1: 0.1, 2: 0.2, 3: 0.3, 4: 0.2, 5: 0.2}
```

# CDF OF DISCRETE RANDOM VARIABLE

It gives the cumulative probability of obtaining a value less than or equal to x.

For example, if you roll a fair 6-sided die, the CDF tells you the probability of rolling a number that is less than or equal to a given value.

If  $X=3$ , the CDF,  $F(3)$ , would be the probability of rolling a 1, 2, or 3.



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** The title bar displays "03-Python-for-Data-Analysis-Pandas".
- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Toolbar:** Includes icons for file operations, search, and various notebook functions.
- Code Cell:** The main area contains the following Python code:

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

# Number of simulations (for approximating the PMF and CDF)
n = 1000

# Simulate a fair die roll (1 to 6)
die_rolls = np.random.randint(1, 7, size=n)

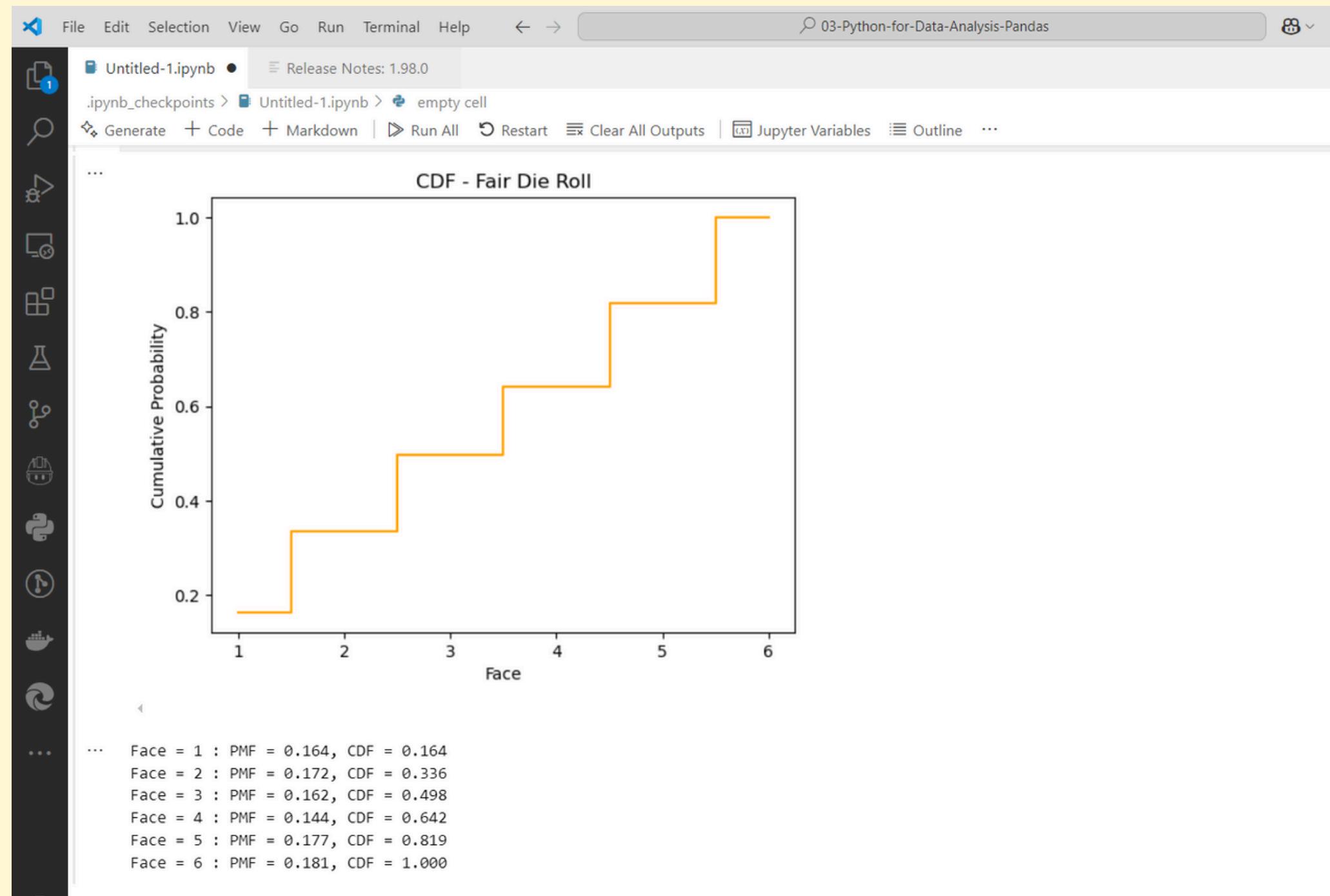
# Calculate PMF (probability mass function)
values, counts = np.unique(die_rolls, return_counts=True)
pmf = counts / n

# Calculate CDF (cumulative probability)
cdf = np.cumsum(pmf)

# Plot the CDF
plt.step(values, cdf, where='mid', color='orange')
plt.title("CDF - Fair Die Roll")
plt.xlabel("Face")
plt.ylabel("Cumulative Probability")
plt.show()

# Print the values for PMF and CDF
for v, p, c in zip(values, pmf, cdf):
    print(f"Face = {v} : PMF = {p:.3f}, CDF = {c:.3f}")
```
- Status Bar:** The bottom status bar shows "[137]" and "0.0s".

# CDF OF DISCRETE RANDOM VARIABLE(CONT)



# CONTINUOUS RANDOM VARIABLE

CONTINUOUS RANDOM VARIABLE: TAKES ON AN INFINITE NUMBER OF POSSIBLE VALUES WITHIN A GIVEN RANGE. FOR EXAMPLE, THE HEIGHT OF A PERSON, WHICH CAN BE ANY REAL NUMBER WITHIN A CERTAIN RANGE.

## A . Uniform distribution

Each element has equal chance to be select.



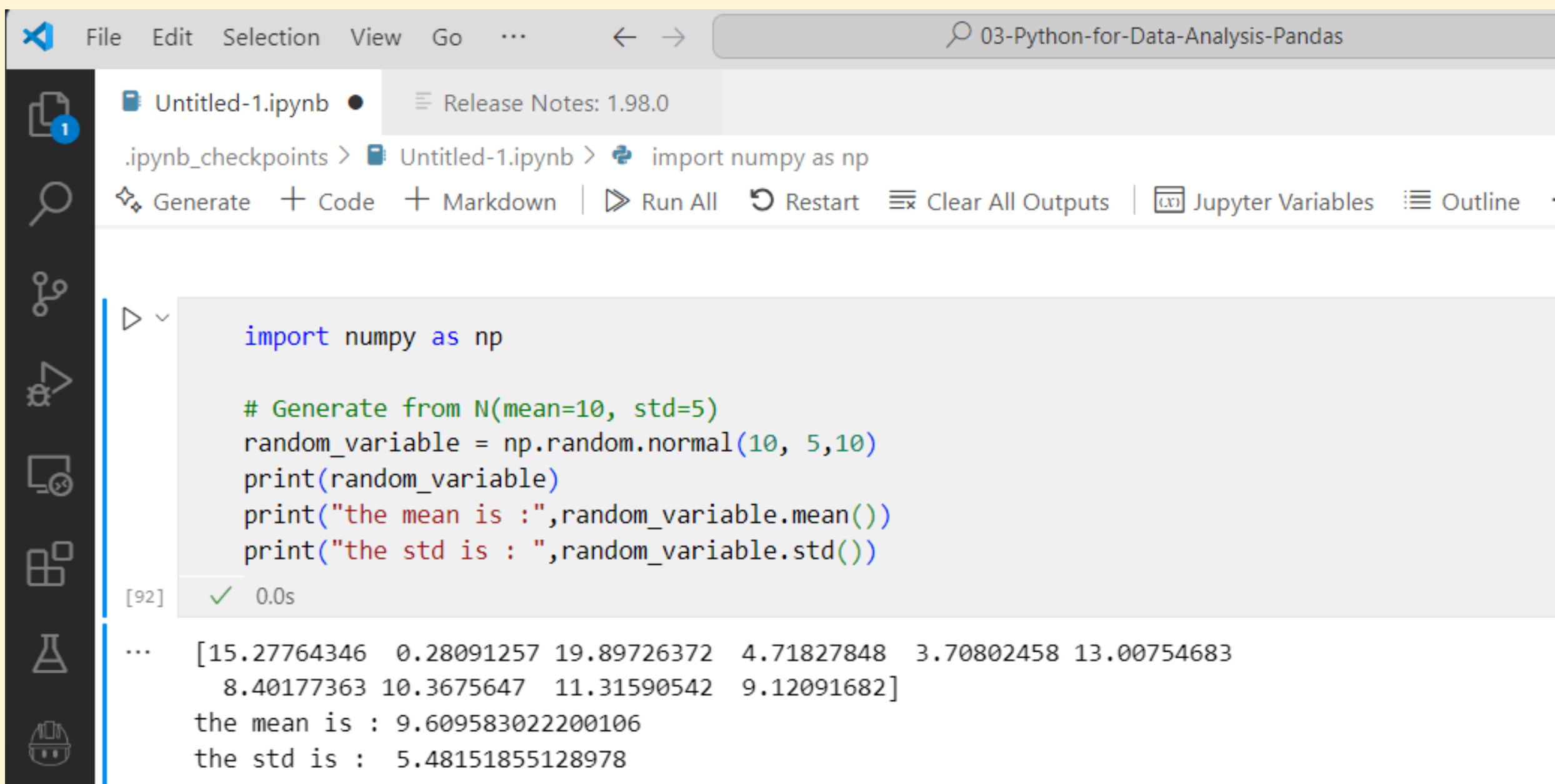
The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** File Edit Selection View Go ... 03-Python-for-Data-Analysis-Pandas
- File List:** Untitled-1.ipynb (1) Release Notes: 1.98.0
- Toolbar:** .ipynb\_checkpoints > Untitled-1.ipynb > import numpy as np
- Buttons:** Generate, + Code, + Markdown, ▶ Run All, ⚡ Restart, Clear All Outputs, Jupyter Variables
- Code Cell:** import numpy as np  
# Generate a random variable between 0 and 1  
random\_variable = np.random.uniform(0, 1)  
print(random\_variable)
- Output Cell:** [61] ✓ 0.0s  
... 0.7369883281315937

# CONTINUOUS RANDOM VARIABLE(CONT)

## B . Normal distribution

Let's generate 10 randoms normal variable with the mean of 10 and the standard deviation of 2 .



The screenshot shows a Jupyter Notebook interface with the following details:

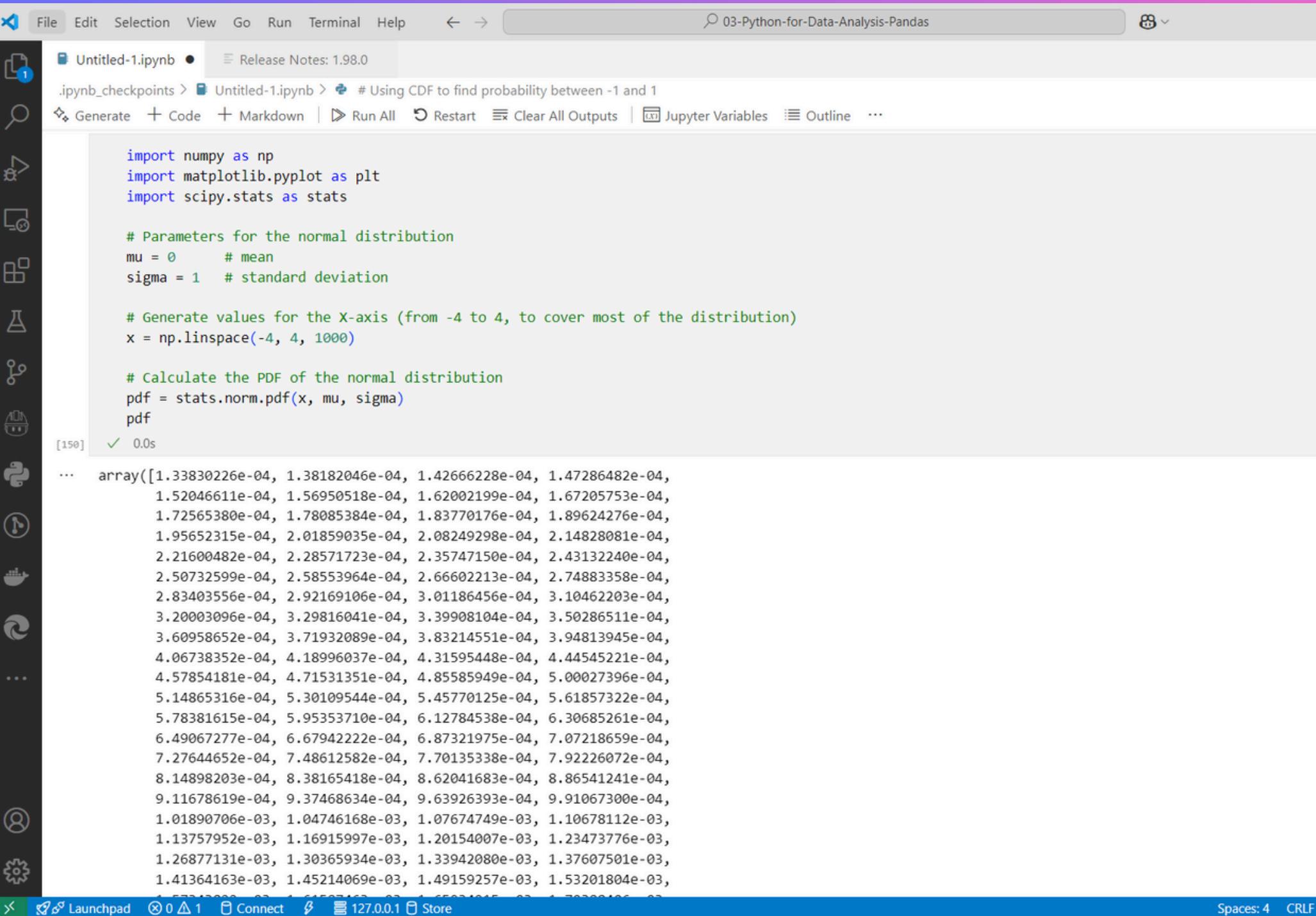
- File Bar:** File, Edit, Selection, View, Go, ...
- Title Bar:** 03-Python-for-Data-Analysis-Pandas
- Toolbar:** Untitled-1.ipynb (1), Release Notes: 1.98.0, .ipynb\_checkpoints, Untitled-1.ipynb, import numpy as np, Generate, + Code, + Markdown, ▶ Run All, ⚡ Restart, Clear All Outputs, Jupyter Variables, Outline, ...
- Code Cell:** [92] import numpy as np  
# Generate from N(mean=10, std=5)  
random\_variable = np.random.normal(10, 5, 10)  
print(random\_variable)  
print("the mean is : ", random\_variable.mean())  
print("the std is : ", random\_variable.std())
- Output Cell:** [92] 0.0s  
... [15.27764346 0.28091257 19.89726372 4.71827848 3.70802458 13.00754683  
8.40177363 10.3675647 11.31590542 9.12091682]  
the mean is : 9.609583022200106  
the std is : 5.48151855128978

# PDF CONTINUOUS RANDOM VARIABLE

## Normal distribution

The Probability Density Function (PDF) is used for continuous random variables. It describes the likelihood of a random variable taking on a specific value.

The PDF gives a density, and the probability of the variable taking on a value in a specific range is calculated by integrating the PDF over that range.



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** File Edit Selection View Go Run Terminal Help
- Search Bar:** 03-Python-for-Data-Analysis-Pandas
- File List:** Untitled-1.ipynb • Release Notes: 1.98.0
- Toolbar:** .ipynb\_checkpoints > Untitled-1.ipynb > # Using CDF to find probability between -1 and 1
- Cell Type:** Code
- Code Content:**

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

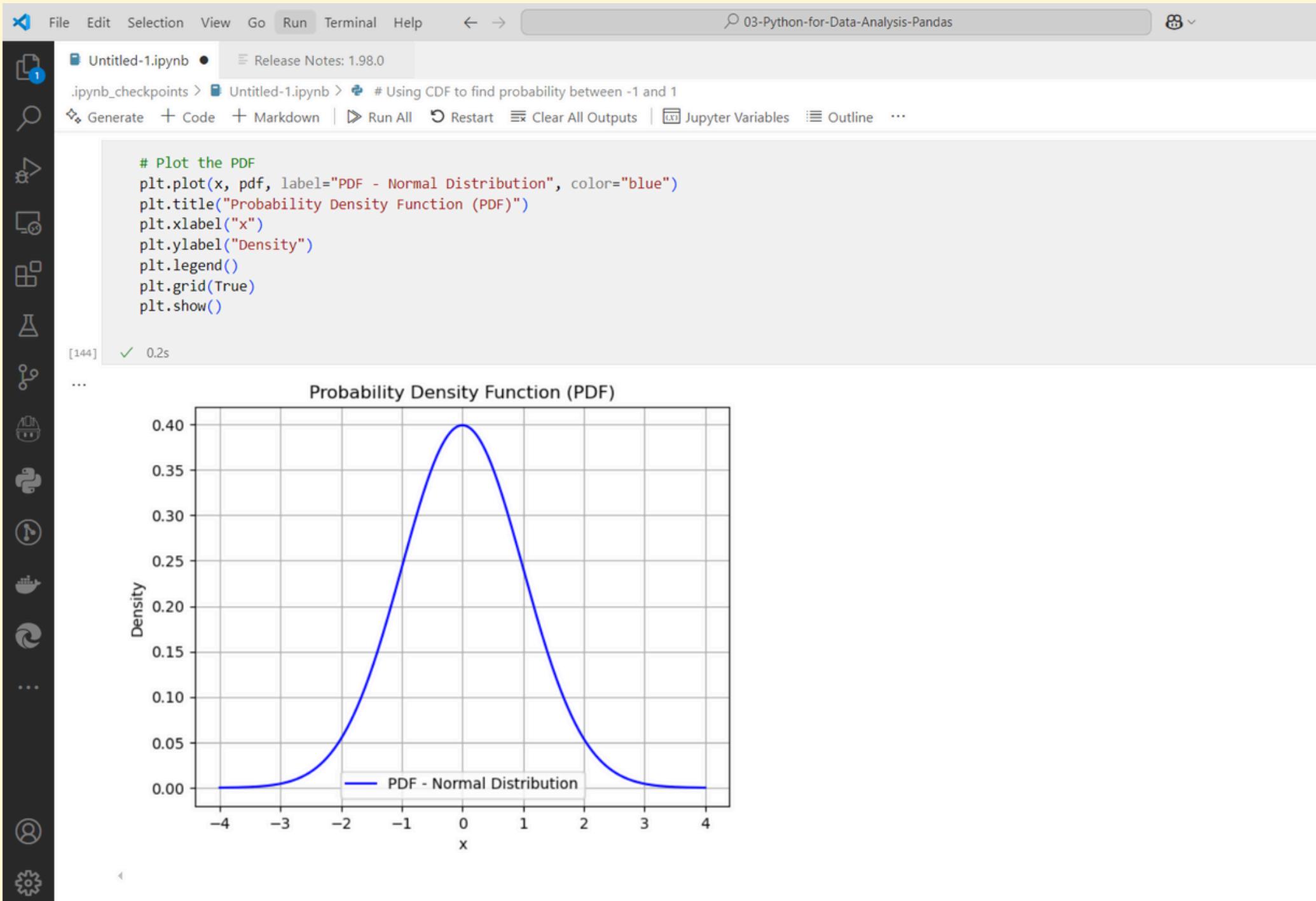
# Parameters for the normal distribution
mu = 0      # mean
sigma = 1    # standard deviation

# Generate values for the X-axis (from -4 to 4, to cover most of the distribution)
x = np.linspace(-4, 4, 1000)

# Calculate the PDF of the normal distribution
pdf = stats.norm.pdf(x, mu, sigma)
pdf
```
- Output:** [150] ✓ 0.0s
- Output Data:** An array of 1000 floating-point numbers representing the PDF values, starting with 1.33830226e-04 and ending with 1.41364163e-03.
- Bottom Bar:** Launchpad, Connect, 127.0.0.1, Store, Spaces: 4, CRLF, Cell

# PDF : THE BELL SHAPEP CURVE OF NORMAL DISTRIBUTION

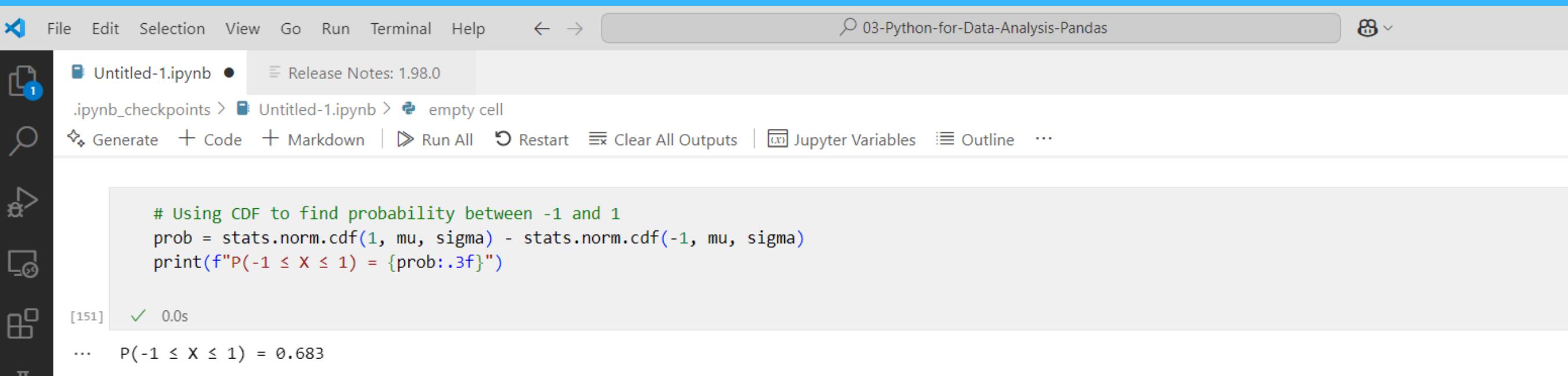
## Normal distribution



# CDF : FOR THE NORMAL DISTRIBUTION

PDF (Probability Density Function) is used for continuous random variables (probability at a specific value is zero, but you calculate probability over ranges).

Let's find the probability between -1 and 1.



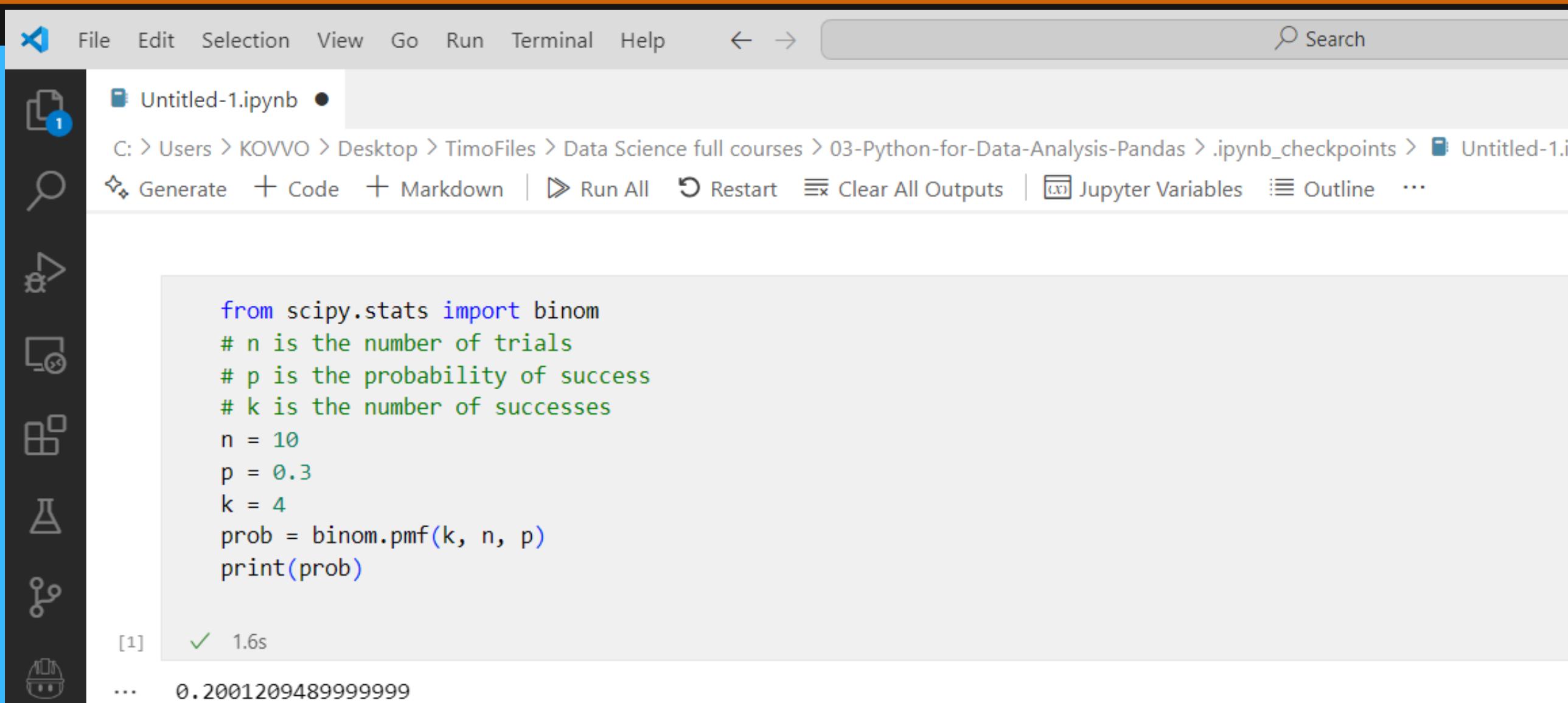
The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** 03-Python-for-Data-Analysis-Pandas.
- Toolbar:** Untitled-1.ipynb (active), Release Notes: 1.98.0, .ipynb\_checkpoints, Untitled-1.ipynb, empty cell, Generate, + Code, + Markdown, Run All, Restart, Clear All Outputs, Jupyter Variables, Outline, ...
- Code Cell:** # Using CDF to find probability between -1 and 1  
prob = stats.norm.cdf(1, mu, sigma) - stats.norm.cdf(-1, mu, sigma)  
print(f"P(-1 ≤ X ≤ 1) = {prob:.3f}")
- Output Cell:** [151] 0.683

# BINOMIAL DISTRIBUTION

The binomial distribution is a discrete probability distribution that models the number of successes in a sequence of  $n$  independent trials, where each trial has a probability  $p$  of success.

Suppose a test has a success probability of 0.3 and is repeated 10 times. The probability of getting exactly 4 successes is:



The screenshot shows a Jupyter Notebook interface with the following details:

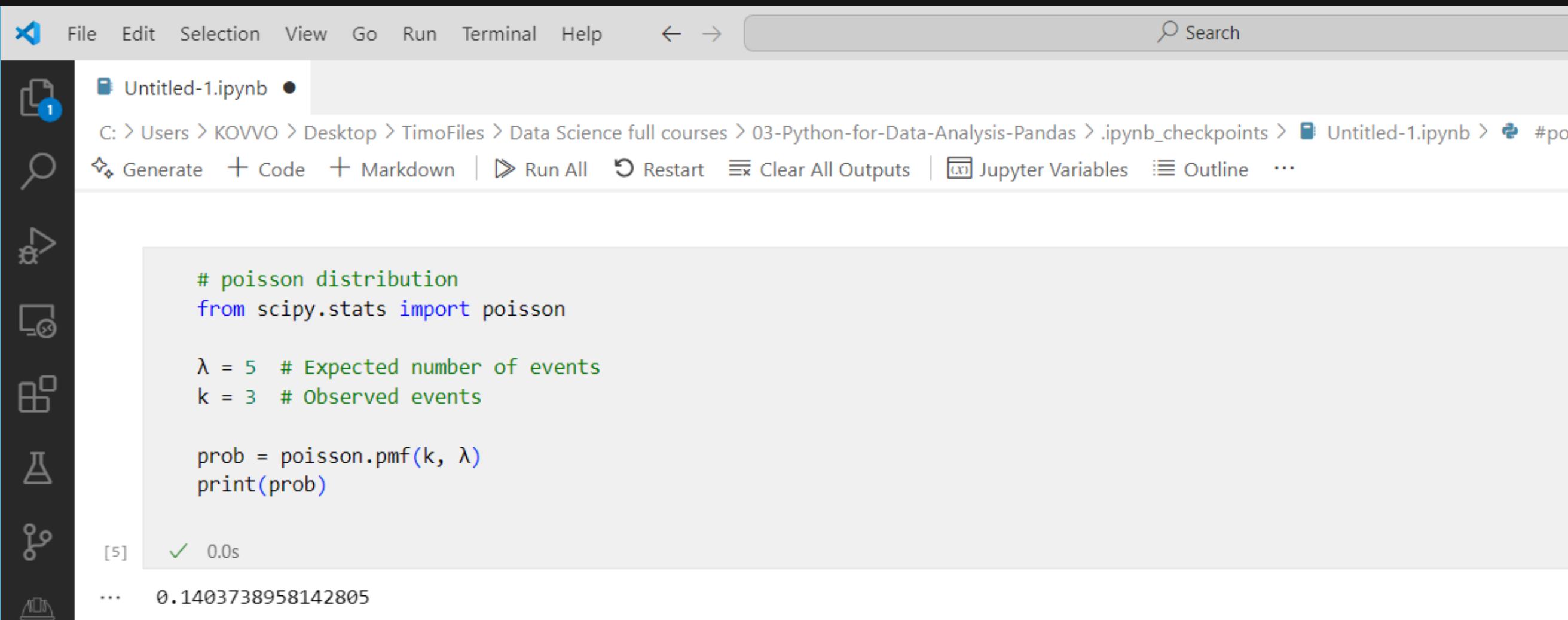
- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** Search.
- Left Sidebar:** A dark sidebar with icons for file operations (New, Open, Save, etc.), search, and other notebook-related functions.
- Toolbar:** Untitled-1.ipynb, C: > Users > KOVVO > Desktop > TimoFiles > Data Science full courses > 03-Python-for-Data-Analysis-Pandas > .ipynb\_checkpoints > Untitled-1.ipynb, Generate, + Code, + Markdown, Run All, Restart, Clear All Outputs, Jupyter Variables, Outline, and a three-dot menu.
- Code Cell:** A code cell containing the following Python script:

```
from scipy.stats import binom
# n is the number of trials
# p is the probability of success
# k is the number of successes
n = 10
p = 0.3
k = 4
prob = binom.pmf(k, n, p)
print(prob)
```
- Output Cell:** [1] 1.6s, followed by the result: ... 0.2001209489999999

# POISSON DISTRIBUTION

The Poisson distribution is a discrete probability distribution that models the number of events occurring in a fixed interval of time or space, given that these events happen independently and at a constant average rate  $\lambda$ .

Suppose a website receives an average of 5 user sign-ups per hour. The probability of getting exactly 3 sign-ups in an hour is:



A screenshot of a Jupyter Notebook interface. The top bar shows the file path: C: > Users > KOVVO > Desktop > TimoFiles > Data Science full courses > 03-Python-for-Data-Analysis-Pandas > .ipynb\_checkpoints > Untitled-1.ipynb. The notebook title is "Untitled-1.ipynb". The toolbar includes icons for Generate, Code, Markdown, Run All, Restart, Clear All Outputs, Jupyter Variables, Outline, and Help. The code cell contains the following Python code:

```
# poisson distribution
from scipy.stats import poisson

λ = 5 # Expected number of events
k = 3 # Observed events

prob = poisson.pmf(k, λ)
print(prob)
```

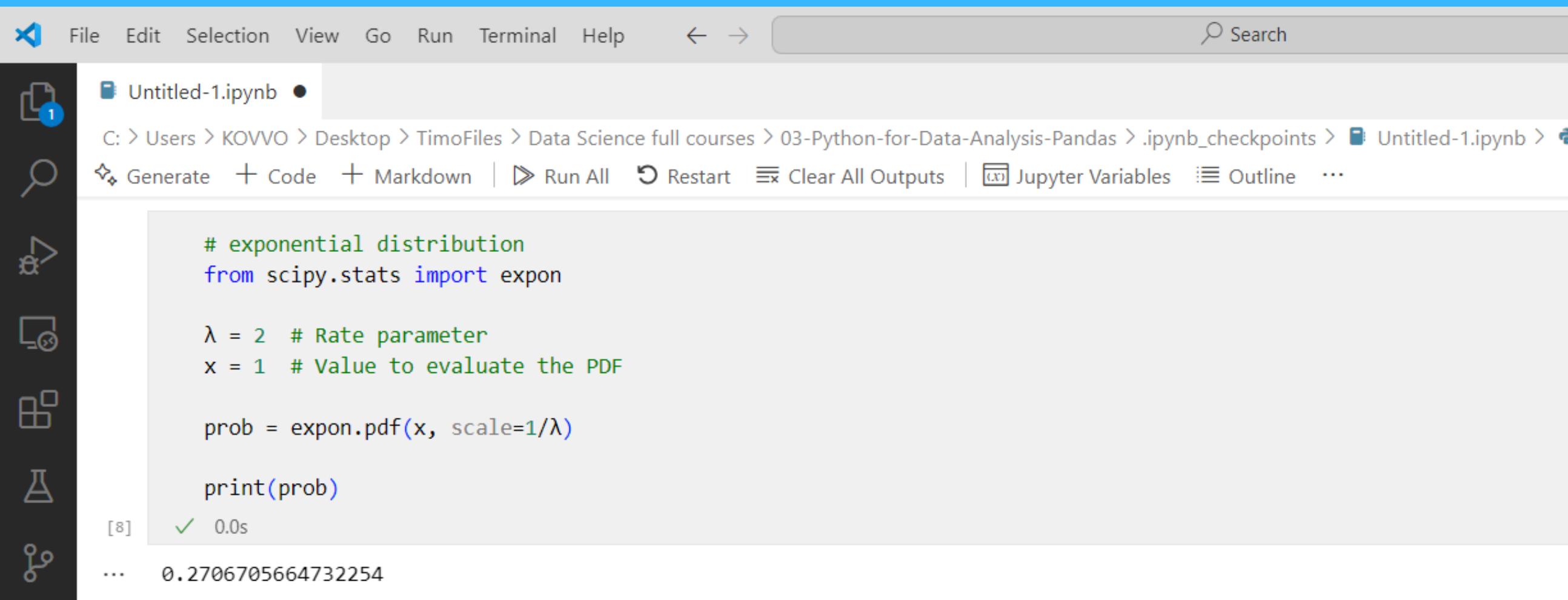
The output cell shows the result of the code execution:

```
[5]    ✓  0.0s
...    0.1403738958142805
```

# EXPONENTIAL DISTRIBUTION

The exponential distribution is a continuous probability distribution that models the time between independent events occurring at a constant average rate  $\lambda$ .

Suppose customer arrivals at a store follow an exponential distribution with an average arrival rate of  $\lambda=2$  customers per hour. The probability that the next customer arrives after more than 1 hour is:



A screenshot of a Jupyter Notebook interface. The top bar shows the menu: File, Edit, Selection, View, Go, Run, Terminal, Help. The search bar is empty. The left sidebar has a file icon with a '1' and 'Untitled-1.ipynb'. Below it are icons for search, cell, and other notebook functions. The main area shows a code cell with the following Python code:

```
# exponential distribution
from scipy.stats import expon

λ = 2 # Rate parameter
x = 1 # Value to evaluate the PDF

prob = expon.pdf(x, scale=1/λ)

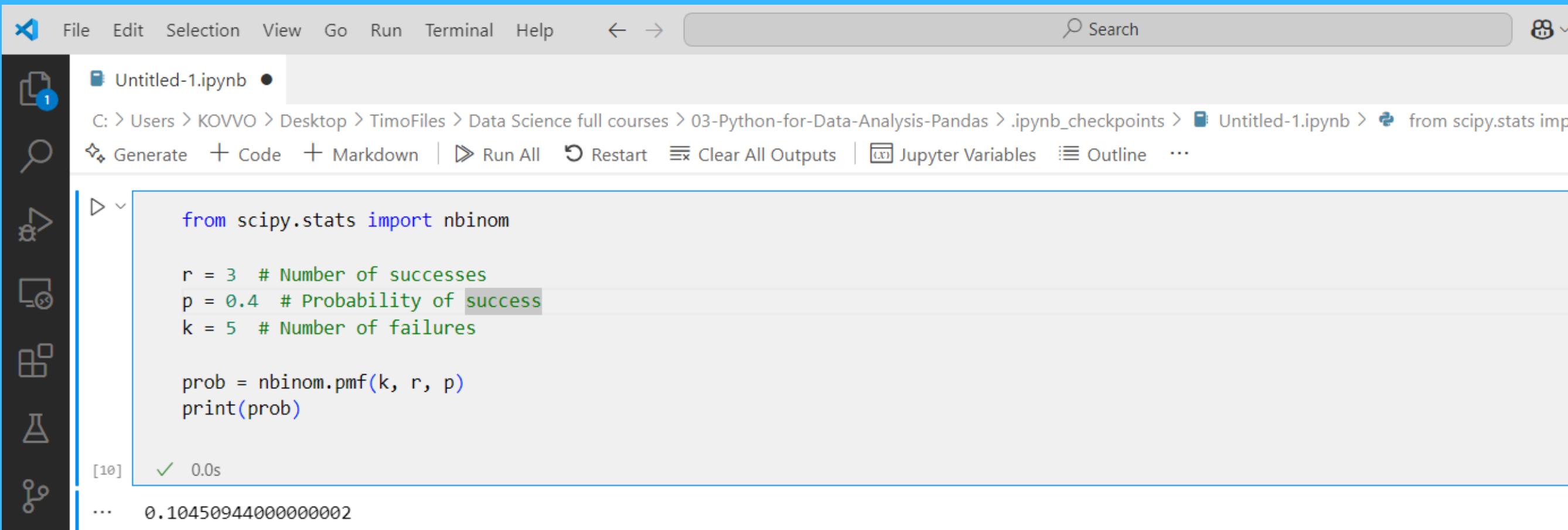
print(prob)
```

The output below the code cell shows two rows: '[8] 0.0s' and '... 0.2706705664732254'. The '...' indicates that the output continues.

# NEGATIVE BINOMIAL DISTRIBUTION

The negative binomial distribution is a discrete probability distribution that models the number of trials required to achieve a fixed number of successes in a sequence of independent Bernoulli trials, where each trial has a probability  $p$  of success.

Suppose a basketball player has a 40% chance ( $p=0.4$ ) of making a shot. What is the probability that they miss 5 times before making 3 successful shots?



A screenshot of a Jupyter Notebook interface. The top bar shows the file path: C: > Users > KOVVO > Desktop > TimoFiles > Data Science full courses > 03-Python-for-Data-Analysis-Pandas > .ipynb\_checkpoints > Untitled-1.ipynb. The notebook title is "Untitled-1.ipynb". The toolbar includes icons for file operations, search, and cell types (Generate, Code, Markdown). Below the toolbar, a code cell contains the following Python code:

```
from scipy.stats import nbinom

r = 3 # Number of successes
p = 0.4 # Probability of success
k = 5 # Number of failures

prob = nbinom.pmf(k, r, p)
print(prob)
```

The output of the cell is:

```
[10]    ✓  0.0s
...    0.10450944000000002
```

---

# THANK YOU



---