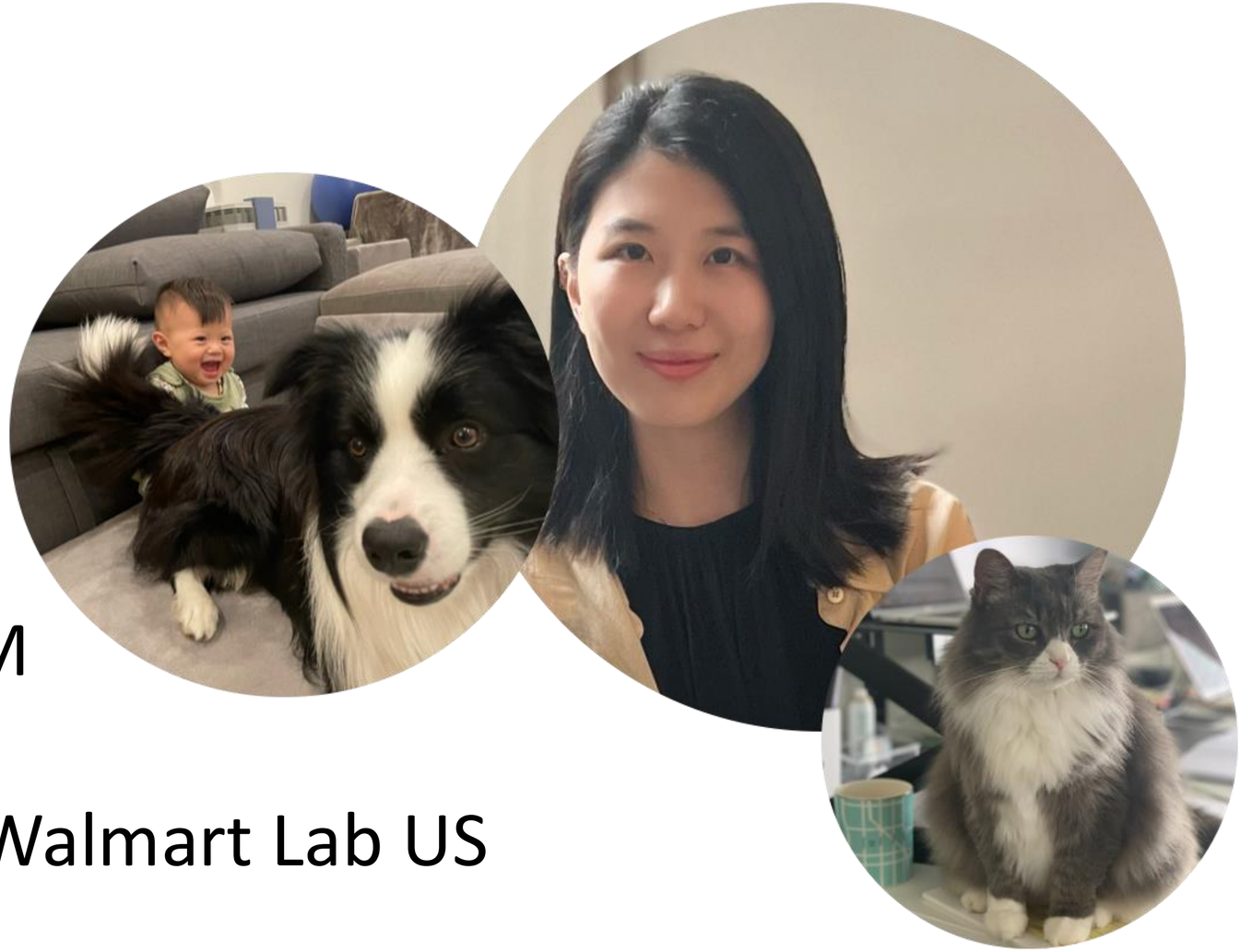
ISOM 2600 – Introduction to Business Analytics

Topic 1: Python Basic
List, Array and Graphing
Readings: Chapter 2, Chapter 4

Hello!

I’m Dr. Xinyu Sun



Lecturer in Department of ISOM

Data Scientist and Manager in Walmart Lab US

Ph.D. in Statistics from Rutgers University

Mom to a 3-year-old boy and two fur babies

Core skills in 2025

1. Analytical thinking
2. Resilience, flexibility and agility
3. Leadership and social influence
4. Creative thinking
5. Motivation and self-awareness
6. Technological literacy
7. Empathy and active listening
8. Curiosity and lifelong learning
9. Talent management
10. Service orientation and customer service

Future of Jobs Report 2025

Top 10 fastest growing skills by 2030

1. AI and big data
2. Networks and cybersecurity
3. Technological literacy
4. Creative thinking
5. Resilience, flexibility and agility
6. Curiosity and lifelong learning
7. Leadership and social influence
8. Talent management
9. Analytical thinking
10. Environmental stewardship

Evolution of Analytics

Knowledge Graphs
"What insights can we derive from data?"

Explainable AI
"Can we trust this prediction?"

Explainable Analytics

Deep Learning
"Can you drive me there?"

Machine Learning
"What can we learn from data?"

Autonomous Analytics

Optimisation
"What's the best that can happen?"

Experimental Design
"What happens if we try this?"

Prescriptive Analytics

Predictive modelling
"What should be recommended?"

Forecasting/Extrapolation
"What if this trend continues?"

Statistical analysis
"Why is this happening?"

Alert
"What actions are needed?"

Query/Drill down
"What exactly is the problem?
When did it happen?
How often?"

Standard reports
"What happened?"

Competitive advantage/Applicability

Sophistication of intelligence

OpenAI Expands Deep Research AI to More Users, Boosting Autonomous Analysis Capabilities

How To Do Data Analysis

Data Analysis is a process of cleaning, transforming, visualizing, and modeling data with the goal of discovering useful patterns

Steps in Data Analysis:

- ❑ Transform raw data into a desired format
- ❑ Clean the data and make necessary modifications
- ❑ Prepare a model and check whether it is overfitting or underfitting
- ❑ Use the model to support business decision-making

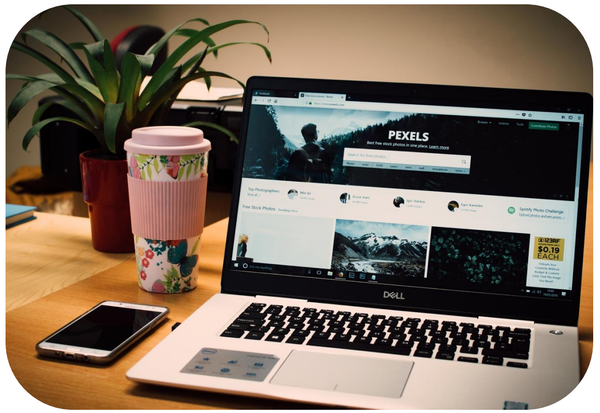
Course Plan

1. Topic 1
   Python Basic
2. Topic 2
   Exploratory Data Analysis
3. Topic 3
   Regression
4. Topic 4
   Clustering

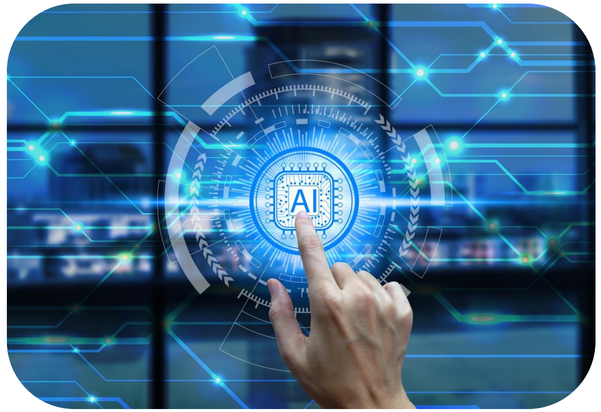
Prerequisites

ISOM 2500 Business Statistics

ISOM 2020 Coding for Business



Laptop with Internet Connection

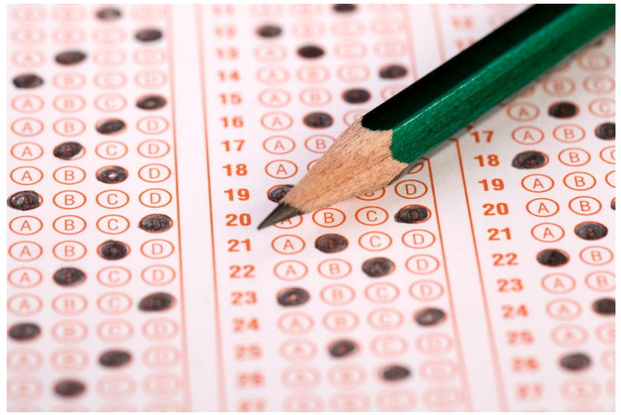


AI Assistant (DeepSeek or ChatGPT)

Mental Readiness

Get Your Hands Dirty

Assessment

Lab Assignments 30%	In-Class Participation 10%	Close-book Final Exam 60%
		
Scheduled by Lab Instructor Start from the week after break (April 7)	**Start today**	**Exam-style exercises will be provided after each class**

Assessment – 2 Lab Assignments

- ❑ 3 persons in each group
- ❑ Soft copy of the assignment submitted via Canvas
- ❑ No Free riding allowed

(Details in Syllabus)

	Contribution to Overall Course Grade	Due Date	Coverage
Assignment 1	15%	Lab instructor will post the announcements	
Assignment 2	15%		

Note: Check Late Submission Policy on Canvas; for all lab assignments related questions, please reach out to lab instructor: Enoch, copied me

Assessment – Participation

- ❑ Complete and submit the **in-class survey** in person on Canvas every class
- ❑ You will earn full participation points (10 points) by participating in **at least 5 out 6 sessions**, 2 points awarded for each session (i.e. you may miss one class)

New: Spot a typo, make suggestions, or give helpful feedback — earn yourself a bag of M&M's!

Assessment – Final Exam

- Individual, 30 **Multiple Choices**, 1-hour exam
- Python will be tested
- Close book, you can bring 2-page A4 size **handwritten** cheat sheets to the exam

Note: If you are absent from the final exam due to illness, you must report the circumstances of the case in writing and provide appropriate evidence or documentation to the instructor and TA within one week of the scheduled date of the final exam. The make-up of the final exam will be provided in the Summer term. Otherwise, if the final exam is missed, you won't pass the course. No make-up final exam will be provided

Office Hours

- Monday 16:00 -17:00 F2F in office LSK4016B or Zoom Meeting by appointment with Dr. Xinyu Sun (imxysun@ust.hk)
- Thursday 15:00 – 17:00 F2F in office LSK4049C or by appointment with Mr. Enoch YIN (imyin@ust.hk)

Email me if you need **immediate help**, please include **“ISOM2600 (L3/L4/L5)”** in the subject line and make sure to include **your name and ID** in email body for easy identification

Course Objectives

Gain understanding of how managers use **business analytics** to formulate and solve business problems and support managerial decision making

Select and apply appropriate **statistical models** in the analysis of quantitative and qualitative data from a variety of business scenarios

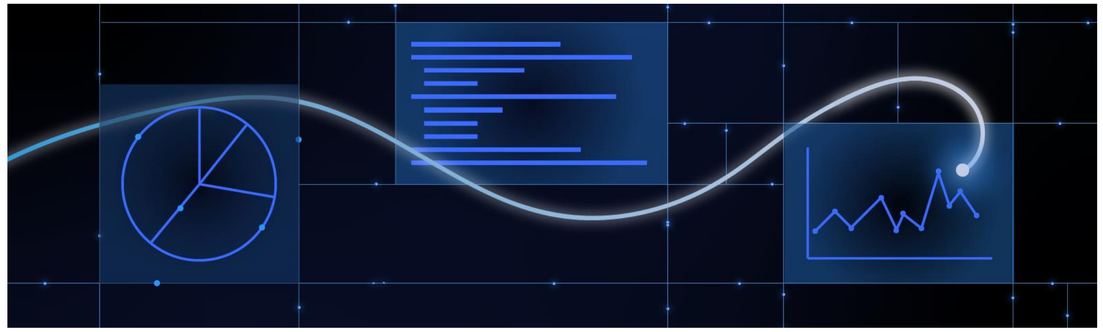
Learn how to use **Python** to apply statistical models on the business problems with real data

Resources

GitHub

kaggle

pandas



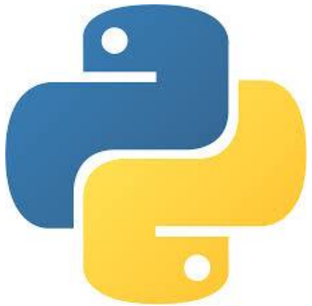
- Materials in Canvas
- Python Data Science Handbook
- Online Tutorials (e.g. Coursera, Library documentations)
- Google Data Science Agent
- Kaggle Cases

Python Basic

Agenda

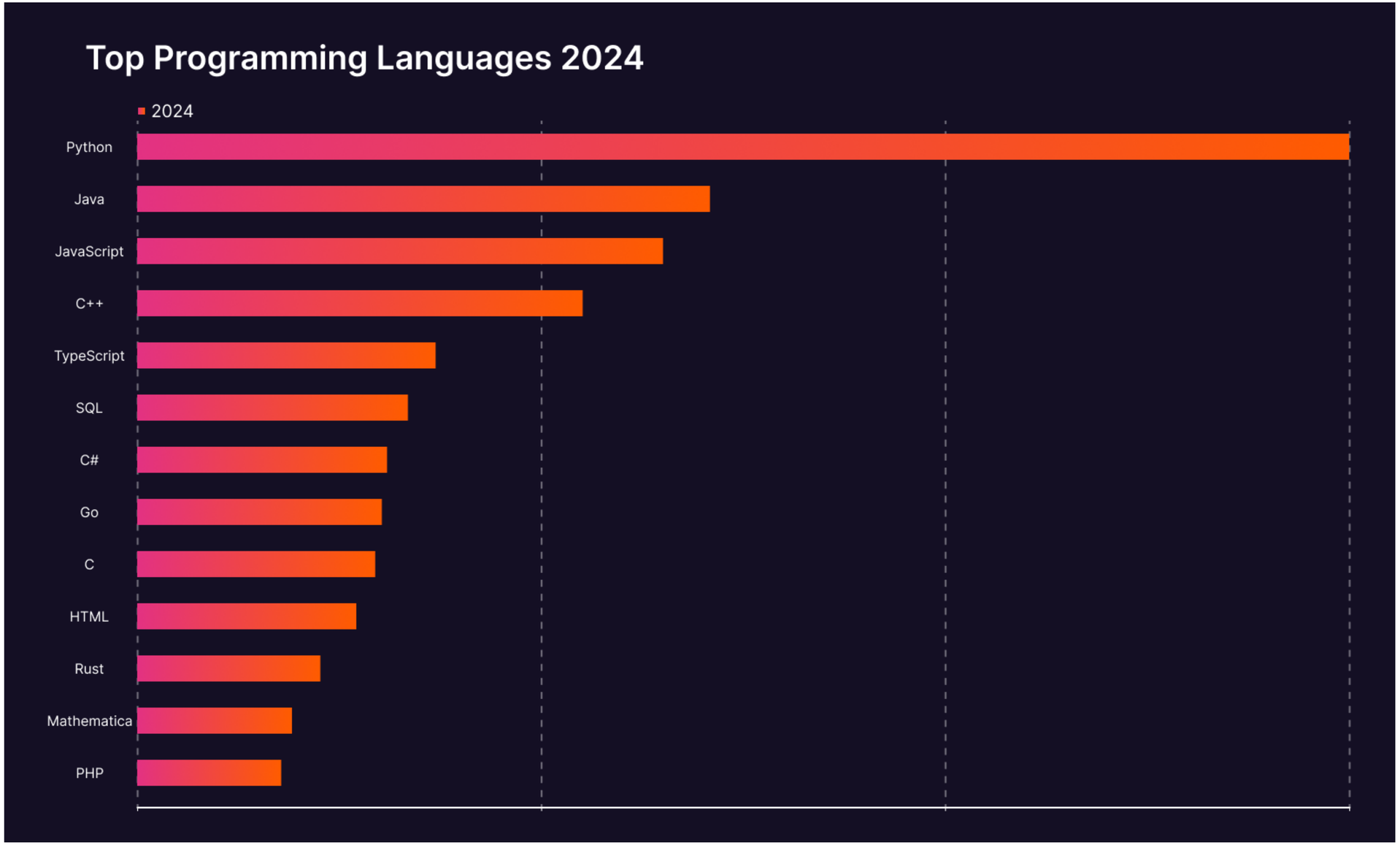
1

Python Basic



- Why Python
- How to use Python
- Review of Basic Data Types
- Data Structures: List and Array
- Introduction of Important Libraries: SciPy and Matplotlib

Why Python



The IEEE Spectrum portal aggregated multiple metrics from various sources to compile the rankings and highlighted key IT trends from recent years.

Why Python

- Easy to Read & Write – Uses English-like syntax, reducing the learning curve
- Modules and Packages – encourage program modularity and code reuse
- Large Community & Support – Thousands of free resources, tutorials, and libraries
- Versatile – Used in business analytics, finance, engineering, and more
- Industry Demand – Python is one of the most in-demand programming languages today

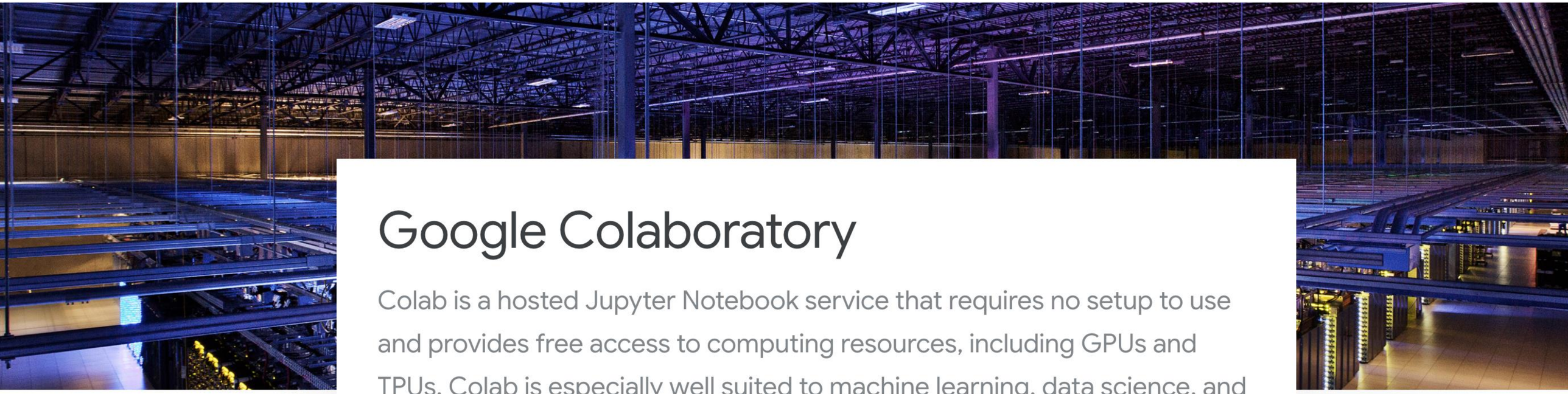
How to Use Python: Editors and IDEs

In this course, we will use **Jupyter Notebook** or **Google Colab**, rather than running Python or IPython in a command line or editors like **PyCharm**

- **Jupyter Notebook** provides an interactive Python interface that runs in a web browser. It does **not require an internet connection** to function
- **Google Colab** allows you to write and execute Python code directly in your browser, offering **free access to GPUs** and seamless collaboration through easy sharing

Why Google Colab

- ❑ Zero Set up Time
- ❑ Easy sharing: share the most updated lecture notes instantly, share your in-class practice
- ❑ You can code in Python and write descriptions as Markdown



Google Colaboratory

Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.

Package and Library

Python's **packages and libraries** make coding easier by providing **pre-built modules and functions** for a wide range of tasks, eliminating the need to write everything from scratch

- ❑**NumPy:** array and matrix, random number generator
- ❑**SciPy:** numerical routines for optimization, statistics
- ❑**Matplotlib:** data visualization

Install Packages

Python environment like Anaconda and Colab come **pre-installed** with many useful packages

Sometimes, we still need to install additional packages, such as plotly, XGBoost (Before installing, you can check if it's already installed)

In **local environment**, you only install them once using **pip**, delete the command after installation from your script

In **Google Colab,** every session starts with a fresh environment. You need to reinstall missing packages every time you restart the notebook

```
# install a package (e.g. Pandas)
!pip install pandas
```

```
# check installed packages
!pip list
```

```
# upgrade a package (e.g. numpy)
!pip install --upgrade numpy
```

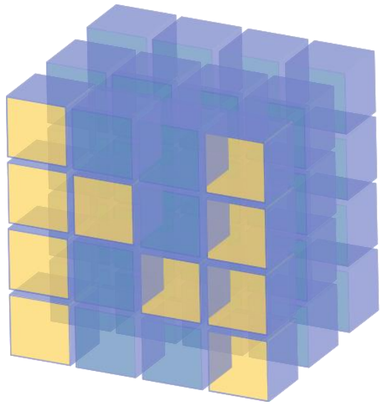
```
# uninstall a package
!pip uninstall seaborn
!pip list
```

Extra: exclamation mark "!" works in Colab and Jupyter Notebooks to execute shell commands, e.g. installing packages

Import Libraries

Once a Python library is installed, you can **import** it into your script or notebook to use its functions and features

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import warnings
warnings.filterwarnings('ignore')
```



NumPy

matplotlib

Extra: Good Python Code Style

Use meaningful variable names

```
count = 10   # ✅ Good
c = 10       # ❌ Bad (not descriptive)
```

Use UPPER_CASE for constants

```
PI = 3.14159
```

Use comments sparingly and meaningfully

```
# Calculate the sum of two numbers
total = a + b
```

Use spaces around operators and after commas

```
x = 10 + 5   # ✅ Good
x=10+5       # ❌ Bad (no spaces)
```

No spaces inside parentheses, brackets, or braces

```
my_list = [1, 2, 3]   # ✅ Good
my_list = [ 1,2,3 ]   # ❌ Bad
```

Python's PEP 8 style guide

Basic Data Types and Data Structures

Case: Understand Data

Dataset URL:
https://www.kaggle.com/datasets/thedevastator/unlock-profits-with-e-commerce-sales-data

SKU	Category	...	currency	Amount	ship-city	ship-state	ship-postal-code	ship-country	promotion-ids	B2B	fulfilled-by
SET389-KR-NP-S	Set	...	INR	647.62	MUMBAI	MAHARASHTRA	400081.0	IN	NaN	False	Easy Ship
JNE3781-KR-XXXL	kurta	...	INR	406.00	BENGALURU	KARNATAKA	560085.0	IN	Amazon PLCC Free-Financing Universal Merchant ...	False	Easy Ship
JNE3371-KR-XL	kurta	...	INR	329.00	NAVI MUMBAI	MAHARASHTRA	410210.0	IN	IN Core Free Shipping 2015/04/08 23-48-5-108	True	NaN
J0341-DR-L	Western Dress	...	INR	753.33	PUDUCHERRY	PUDUCHERRY	605008.0	IN	NaN	False	Easy Ship
JNE3671-TU-XXXL	Top	...	INR	574.00	CHENNAI	TAMIL NADU	600073.0	IN	NaN	False	NaN

Int & Float

Number is used to store numeric values

Numeric type:

- ❑ int: for signed integers
- ❑ float: for floating-point/decimal numbers

```
# Int: Whole numbers (positive, negative, or zero)
x = 10    # Integer
y = -5    # Negative integer
z = 0     # Zero
print(type(x), type(y), type(z))
```

```
<class 'int'> <class 'int'> <class 'int'>
```

```
# Float: Decimal numbers
a = 3.14   # Float
b = -0.5   # Negative float
c = 2.0    # Float (even though it's a whole number)
print(type(a), type(b), type(c))
```

```
<class 'float'> <class 'float'> <class 'float'>
```

Bool

A Boolean value is either true or false. It is name after the British mathematician, **George Boole**, who first formulated Boolean algebra – some rules for reasoning about and combing these values

```
type(True),type(False)
```

```
(bool, bool)
```

```
a=10==20
a,type(a)
```

```
(False, bool)
```

```
x = 10
y = 5
print(x > y)
print(x == y)
```

```
True
False
```

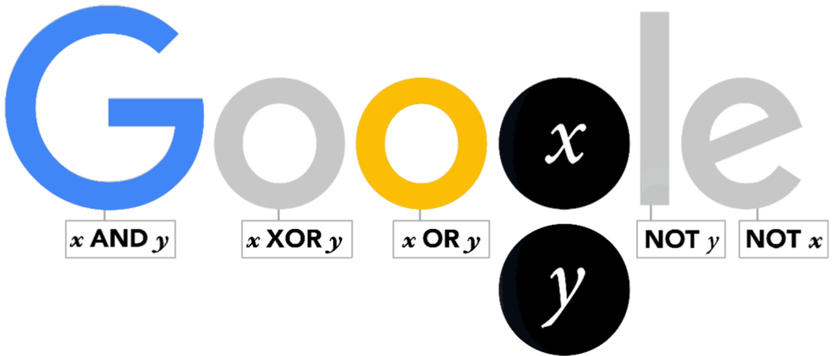
In Python, the two Boolean values are **True** and **False** (with exact **capitalization** as shown), and their corresponding **data type is bool**. True is equivalent to 1, while False is equivalent to 0

```
alist=[1,2,3,4,5]
blist=[x>3 for x in alist]
print(blist)
```

```
[False, False, False, True, True]
```

```
sum(blist)
```

```
2
```

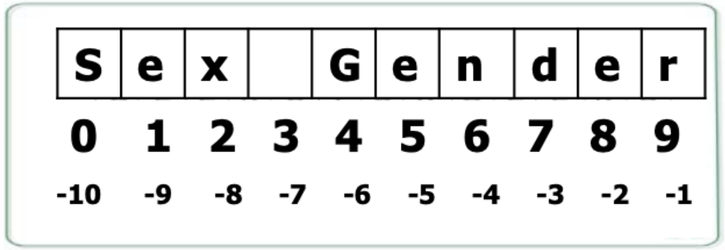


String

- A **string (**str**)** in Python is a sequence of characters enclosed in **single quotes (')**, **double quotes (")**, or **triple quotes (''')**.
- Strings are **immutable**, meaning their contents cannot be changed after creation
- Use **slice[:]** (square brackets with colons and integers inside) operators to access the data of string: index starts at 0; don't include the stopping point
- The operator + is used to **concatenate** two strings

```
firstVariable="Sex Gender"
print(firstVariable)
```

```
Sex Gender
```




```
firstVariable[0:4]
```

```
'Sex '
```

```
firstVariable[-2:-1]
```

```
'e'
```

```
firstVariable[:3] + ' ' + firstVariable[-6:]
```

```
'Sex Gender'
```

Game: String Spy Mission

1. Get the first word of the headline
2. Slice out the acronym ‘GPT’ from the headline
3. Concatenate OpenAI and GPT to get ”Open AI GPT”

Extra: Everything is an Object in Python

Each object has **attributes** and **methods**

- Attributes: store information (data) about the object

object.attribute

```
aNumber = 100
aNumber.bit_length() #method of the integer type
```

```
7
```

```
aNumber.real #attribute of the integer type
```

```
100
```

- Methods: define behavior (functions)

object.method()

```
text = "hello world"
text.upper() #method: Converts string to uppercase
```

```
'HELLO WORLD'
```

In-class Exercise 1

1. Input Your Last Name
2. Print “Mr/Ms Last Name”

```
myLast=input("What is your Last Name?")
print("Hello Mr/Ms "+myLast)
```

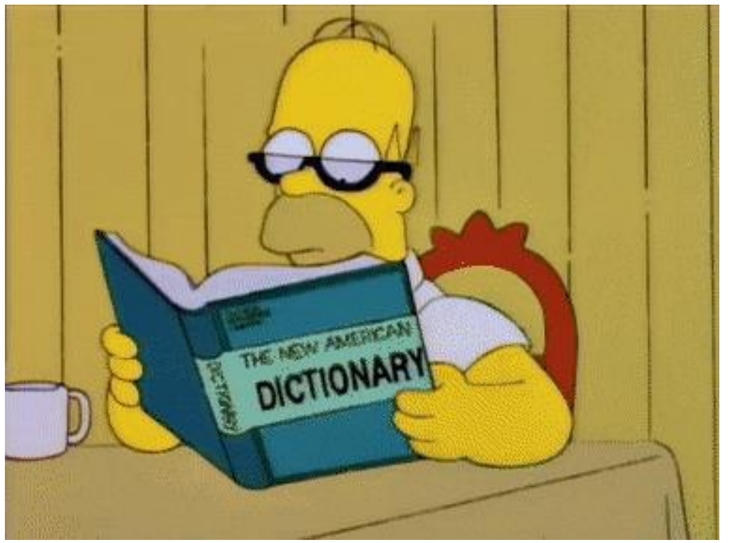
```
What is your Last Name?Sun
Hello Mr/Ms Sun
```

Extra: Data Structures in Python

Python provides **built-in data structures** like **stacks, queues, and dictionaries** for efficient data management

Stack: LIFO (Last In, First Out) best for situations where the most recently added data needs to be accessed first

Queue: FIFO (First In, First Out)



A **dictionary** (dict) stores data in key-value pairs, allowing fast lookups

Extra: Data Structures in Python

An **array** is a collection of elements of the **same type**, used for efficient **numerical** computations

A **list** in Python is an **ordered, mutable** collection that can hold **different types** of elements

List: A Universal Data Container

Items in a list are **separated by commas (,)** and enclosed within **square brackets** []

Lists are used to **store multiple items of mixed different types** (int, float, string, boolean, etc.) in a single variable

Lists are ordered, can be accessed using **indexing and slicing**

Lists are **mutable**, have powerful **built-in methods** like append(), remove()…

```
my_list = [10, 3.14, "Hello", True]
my_list[0], my_list[-1]
```

```
(10, True)
```

```
my_list.append("Python")
my_list
```

```
[10, 3.14, 'Hello', True, 'Python']
```

```
my_list.remove("Hello")
my_list
```

```
[10, 3.14, True, 'Python']
```

List Comprehension

List comprehension is a concise and efficient way to **create lists** in Python

It allows you to generate a **new list** by applying **an expression** to each element in **an iterable** (like a list, range, or tuple) in just one line of code

Extra: List comprehension is **faster** than traditional loops because Python optimizes its execution ***(Code on the right is extra)***

```
import time
```

```
# using for loop
size = 10**3
alist=[i for i in range(size)]
newlist=[]
start_time_for = time.time()
for x in alist:
  newlist.append(x**2)
end_time_for = time.time()
time_for = end_time_for - start_time_for
print(f"Time taken by for loop: {time_for:.6f} seconds")
```

```
Time taken by for loop: 0.000561 seconds
```

```
# make a new list for every item of the orginal list
start_time_lc = time.time()
ylist=[x**2 for x in alist]
end_time_lc = time.time()
time_lc = end_time_lc - start_time_lc
print(f"Time taken by list comprehension: {time_lc:.6f} seconds")
```

```
Time taken by list comprehension: 0.000216 seconds
```

```
time_for > time_lc
```

```
True
```

List Comprehension

Plain Comprehension: new_list = [**expression** for **member** in **iterable**]

```
numbers = [i for i in range(10)]
print(numbers)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
xlist=[1,2,3,4,5,100,200]
ylist=[x**2 for x in xlist]
print(ylist)
```

```
[1, 4, 9, 16, 25, 10000, 40000]
```

Filtered Comprehension: new_list = **[expression for member in iterable (if conditional)]**
Conditional Comprehension: new_list **= [expression (if conditional) for member in iterable]**

```
xlist=[1,2,3,4,5,100,200]
zlist=[x for x in xlist if x >50]
print(zlist)
```

```
[100, 200]
```

```
xlist=[1,2,3,4,5,100,200]
hlist=[x**2 if x<50 else x**0.5 for x in xlist]
print(hlist)
```

```
[1, 4, 9, 16, 25, 10.0, 14.142135623730951]
```

Array: Numpy ndarray

NumPy library gives Python the ability to efficiently handle **arrays** and **matrices**

It provides a high-performance scientific computation tools working with multi-dimensional arrays (1D, 2D, 3D+)

- ❑ The array object in NumPy is called ndarray, it can be defined from **a list** (contain the same type, if not, **upcast** if possible)
- ❑ It can do **element-wise** computation

```
import numpy as np
firstarray=np.array([1,2,3.14,4])
firstarray
```

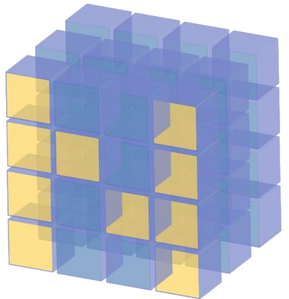
```
array([1.  , 2.  , 3.14, 4.  ])
```

```
firstarray + 3
```

```
array([4.  , 5.  , 6.14, 7.  ])
```

```
firstarray * 2
```

```
array([2.  , 4.  , 6.28, 8.  ])
```



NumPy

Extra: Understanding Datasets as Arrays of Numbers

Datasets come in various formats—text, images, audio, and numerical data—but can all be represented as **arrays of numbers**:

Text: converted to numerical form using word embeddings (e.g., Word2Vec, TF-IDF).

Images: represented as pixel intensity values in multi-dimensional arrays.

Audio: stored as numerical waveforms or spectrograms.

Structured Data: stored as tabular numerical values in spreadsheets or databases.

Thinking of data this way enables **efficient processing, analysis, and machine learning applications** across all types

Extra: Image as Array


```
Image Array Shape: (442, 600, 3)
```

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image in color mode
image_path = "UST_C3.jpg"
image = cv2.imread(image_path)  # Reads image as a NumPy array (B

# Convert BGR to RGB (OpenCV loads images in BGR order)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Convert to NumPy array (if not already)
image_array = np.array(image_rgb)

# Display the image
plt.imshow(image_array)
plt.axis("off")  # Hide axes
plt.title("Colorful Image as Array")
plt.show()

# Print a small portion of the array
print("Image Array Shape:", image_array.shape)
print("Image Array (first 2x2 pixels):")
print(image_array[:2, :2])  # Display pixel values of top-left co
```

Multidimensional Array

Create 2D and 3D arrays

```
arr2D = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr2D)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
arr3D = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print(arr3D)
```

```
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
```

Access **one item** using a comma-separated tuples of **indices**

```
arr2D[1,2]
```

```
6
```

```
arr3D[1,1,1]
```

```
8
```

Example: Create Arrays

Built-in functions to create arrays

```
np.zeros((3,3))
```

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

```
np.ones((3,3))
```

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

```
np.eye(3)
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
# Create an array of five values evenly spaced between 0 and 1
np.linspace(0,1,5)
```

```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

```
# Create an array filled with a linear sequence
# Starting at 3, ending at 10, stepping by 2
np.arange(3,10,2)
```

```
array([3, 5, 7, 9])
```

Array creation routines based on numerical ranges. It return evenly spaced values within a given interval

Array: Random Number Generators

NumPy provides a powerful **random number generation** module through **numpy.random.Generator** to generate random numbers from different distributions, e.g.

- ❑ Standard normal
- ❑ Normal
- ❑ Uniform

```
# 2*3 array with samples from standard normal
A = np.random.randn(2, 3)
print(A)
```

```
[[ 0.49587473 -0.09535931  0.790355  ]
 [-1.3121041   0.09631029 -0.30301843]]
```

```
# 2*3 array with samples from normal with mean = 10 and sd= 20
B = np.random.normal(10, 20, (2, 3))
print(B)
```

```
[[-15.86724955  24.66869156  22.52122657]
 [  0.87494479  14.6507978   31.10191167]]
```

```
# 2*3 array with samples from uniform[0,1]
C = np.random.rand(2, 3)
print(C)
```

```
[[0.34360002 0.77019023 0.86727236]
 [0.24009241 0.02458075 0.8416512 ]]
```

Index of Array and Slicing

In NumPy, **square brackets** with a comma-separated tuple of indices can be used to access **subarrays** using slice notation (:)

Multidimensional slicing works the same way, with multiple slices separated by commas ***(Don't include the stopping point)***

```
D = np.random.randint(0, 10, (5, 5))
print(D)
```

```
[[1 4 7 8 2]
 [3 2 7 8 9]
 [0 8 5 5 1]
 [9 0 4 6 1]
 [4 7 6 1 5]]
```

```
D[4,:]
```

```
array([4, 7, 6, 1, 5])
```

```
D[:2,:3]
```

```
array([[1, 4, 7],
       [3, 2, 7]])
```

Operation with Array

Element-wise Computation

```
xArray = np.array([[1, 2], [3, 4]])
xArray
```

```
array([[1, 2],
       [3, 4]])
```

```
yArray = np.ones((2,2))
yArray
```

```
array([[1., 1.],
       [1., 1.]])
```

```
xArray + yArray
```

```
array([[2., 3.],
       [4., 5.]])
```

```
xArray - yArray
```

```
array([[0., 1.],
       [2., 3.]])
```

```
xArray * yArray
```

```
array([[1., 2.],
       [3., 4.]])
```

```
xArray / yArray
```

```
array([[1., 2.],
       [3., 4.]])
```

Operation with Array

Aggregation: Statistics Methods

```
data = np.array([[10, 20, 30, 40], [50, 60, 70, 80]])
print(data)
print(np.mean(data))
print(np.median(data))
print(np.min(data))
print(np.max(data))
print(np.std(data))
print(np.sum(data))
print(np.cumsum(data))
```

```
[[10 20 30 40]
 [50 60 70 80]]
45.0
45.0
10
80
22.9128784747792
360
[ 10  30  60 100 150 210 280 360]
```

```
print(data.mean())
print(data.min())
print(data.max())
print(data.std())
print(data.sum())
print(data.cumsum())
```

```
45.0
10
80
22.9128784747792
360
[ 10  30  60 100 150 210 280 360]
```

```
print(np.percentile(data, 25))
print(np.median(data))
print(np.percentile(data, 75))
```

```
27.5
45.0
62.5
```

Operation with Array

Aggregation

By default, each NumPy aggregation function will return the aggregate over the entire Array or take an additional argument specifying the **axis** along which the aggregate is computed

```
print(data.sum())
print(data.sum(axis = 0)) # by column
print(data.sum(axis = 1)) # by row
```

```
360
[ 60  80 100 120]
[100 260]
```

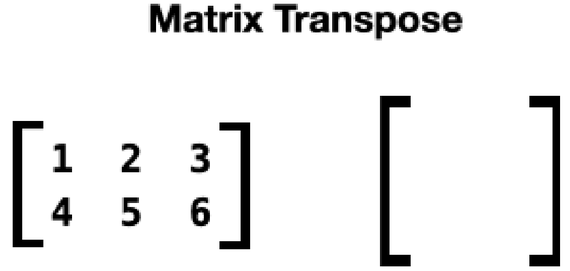
```
print(data.max())
print(data.max(axis = 0)) # by column
print(data.max(axis = 1)) # by row
```

```
80
[50 60 70 80]
[40 80]
```

Extra: More Operation with Array

Transpose

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$




```
A = np.array([[1, 2], [3, 4], [5, 6]])
print(A)
print(A.T)
```

```
[[1 2]
 [3 4]
 [5 6]]
[[1 3 5]
 [2 4 6]]
```

```
B = np.array([[1, 2, 3], [4, 5, 6]])
print(B)
print(B.T)
```

```
[[1 2 3]
 [4 5 6]]
[[1 4]
 [2 5]
 [3 6]]
```

In-class Exercise 2

What are the 5-number summary statistics of the US Presidents?

Sample data:

	Name	"Height (inches)"
0	George Washington	74
1	John Adams	67
2	Thomas Jefferson	75
3	James Madison	64
4	James Monroe	72

```
print("Mean height: ", heights.mean())
print("Standard deviation:", heights.std())
print("Minimum height: ", heights.min())
print("Maximum height: ", heights.max())
print("25th percentile: ", np.percentile(heights, 25))
print("Median: ", np.median(heights))
print("75th percentile: ", np.percentile(heights, 75))
```

```
Mean height:  70.8
Standard deviation: 2.729265265394496
Minimum height:  64
Maximum height:  75
25th percentile:  69.0
Median:  71.0
75th percentile:  73.0
```

SciPy

SciPy (Scientific Python) is a powerful **open-source Python library** built on **NumPy**, designed for **scientific computing, mathematics, and engineering**

Features	SciPy Functionality
Optimization	scipy.optimize
Linear Algebra	scipy.linalg
Integration	scipy.integrate
Signal Processing	scipy.signal
Statistics	scipy.stats


```
import scipy.stats
```

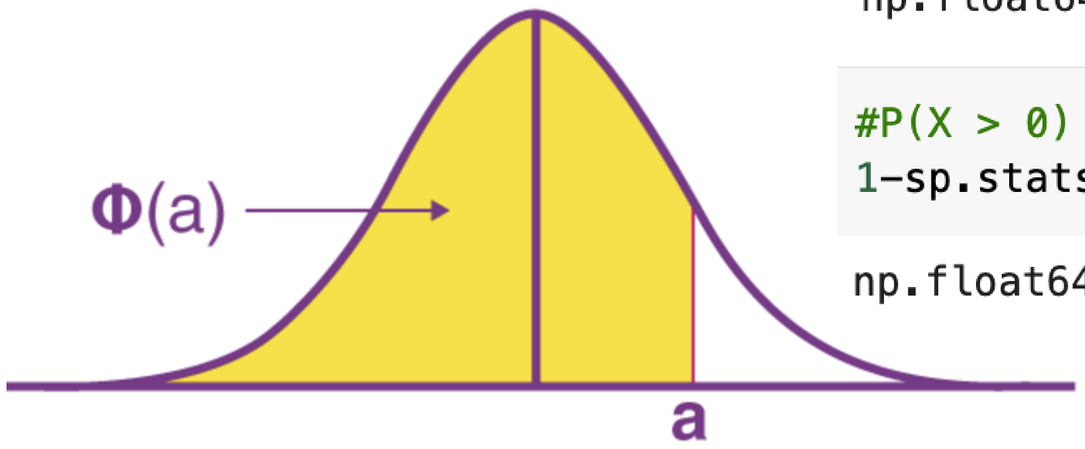
Statistics in SciPy

Example: Calculate Normal Probability

Compute Cumulative Distribution Function (CDF) of Normal Distribution

$\Phi(a) = P(X < a)$

$P(X \geq a) = 1 - \Phi(a)$




```
#P(X < 0) = ?  X is normal with mean 8 and sd 5
# a = 0, calculate prob from negative infinity to this value
sp.stats.norm.cdf(0,loc = 8, scale = 5)
```

```
np.float64(0.05479929169955797 4)
```

```
#P(X > 0) = ?  X is normal with mean 8 and sd 5
1-sp.stats.norm.cdf(0,8,5)
```

```
np.float64(0.945200708300442)
```

Statistics in SciPy

Example: Find Normal Critical Value

```
# Find the uppper 95th percentile of standard normal
# P(X < ?) = 0.95 X is standard normal with mean 0 and sd 1
sp.stats.norm.ppf(0.95,loc = 0, scale = 1)
```

```
np.float64(1.6448536269514722)
```

```
# Find the upper 5th percentile of standard normal
# P(X < ?) = 0.05 X is standard normal with mean 0 and sd 1
sp.stats.norm.ppf(0.05,loc = 0, scale = 1)
```

```
np.float64(-1.6448536269514729)
```

In-Class Exercise 3

We have a Hypothesis Testing question:

$H_0: \mu \leq 100$ v.s. $H_a: \mu > 100$

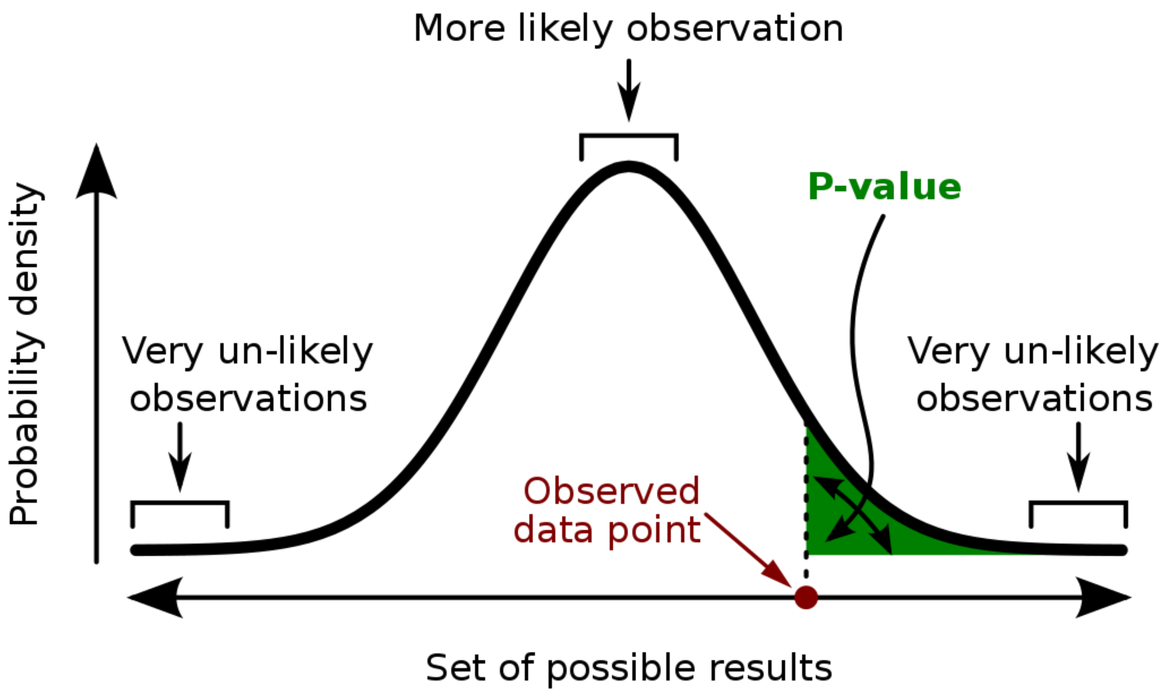
If we know the test statistic

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}} = 2.3$$

What is the P-value

$$P(Z \geq z | H_0 \, is \, true)$$

where Z is a standard normal?



A **p-value** (shaded green area) is the probability of an observed (or more extreme) result assuming that the null hypothesis is true.

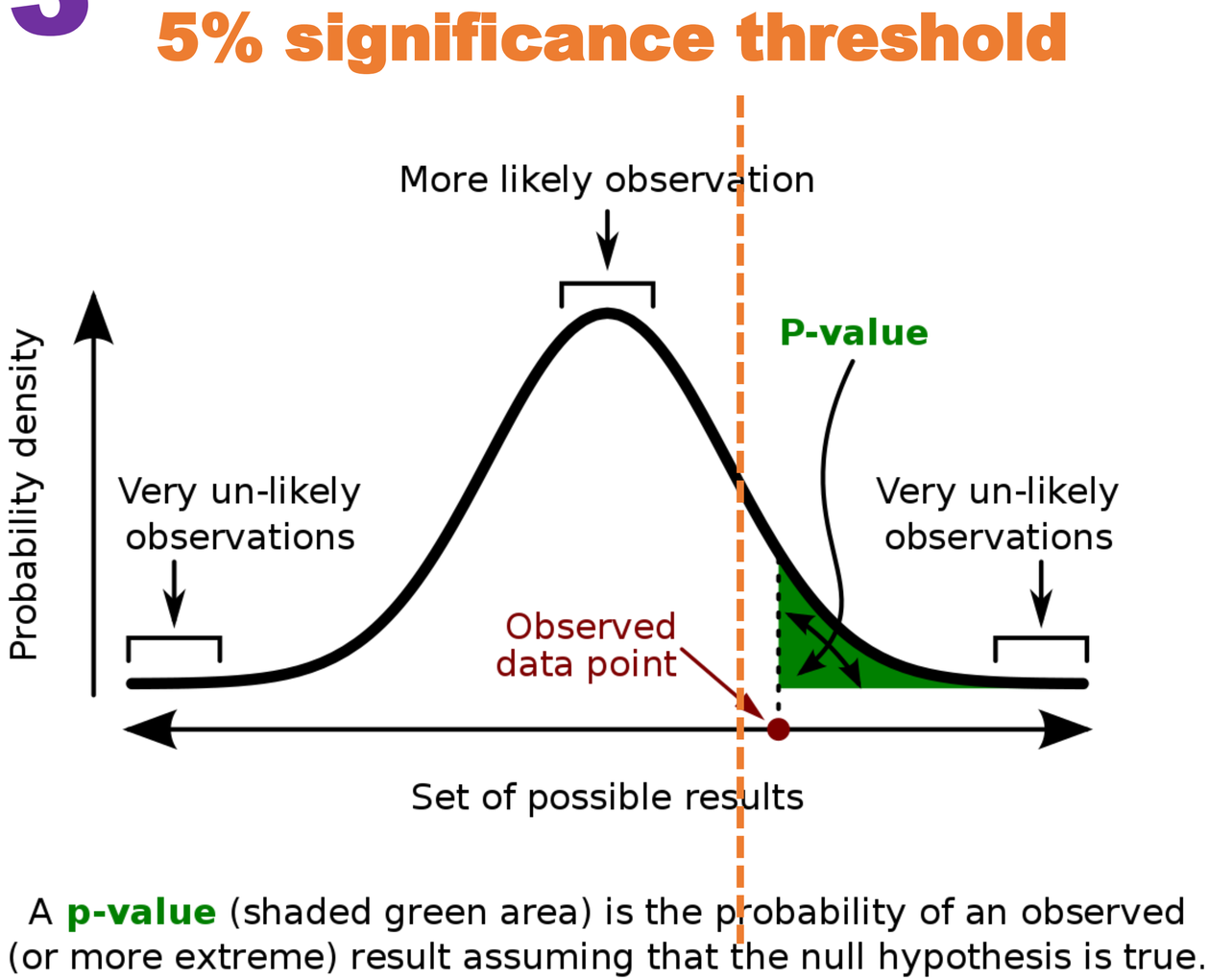
In-Class Exercise 3

We compare the **P-value** with the **significance value** to make the decision

If P-Value < Significance Level = 5%, then **reject Null**

```
pvalue = 1 - sp.stats.norm.cdf(2.3,loc = 0, scale = 1)
print("The p-value is ", pvalue)
```

```
The p-value is  0.010724110021675837
```



Statistical Graphing

matplotlib

seaborn

```
import matplotlib.pyplot as plt
```

Data Visualization

Matplotlib is a **multiplatform data visualization library** built on **NumPy arrays** and designed to integrate seamlessly with the broader **SciPy stack**

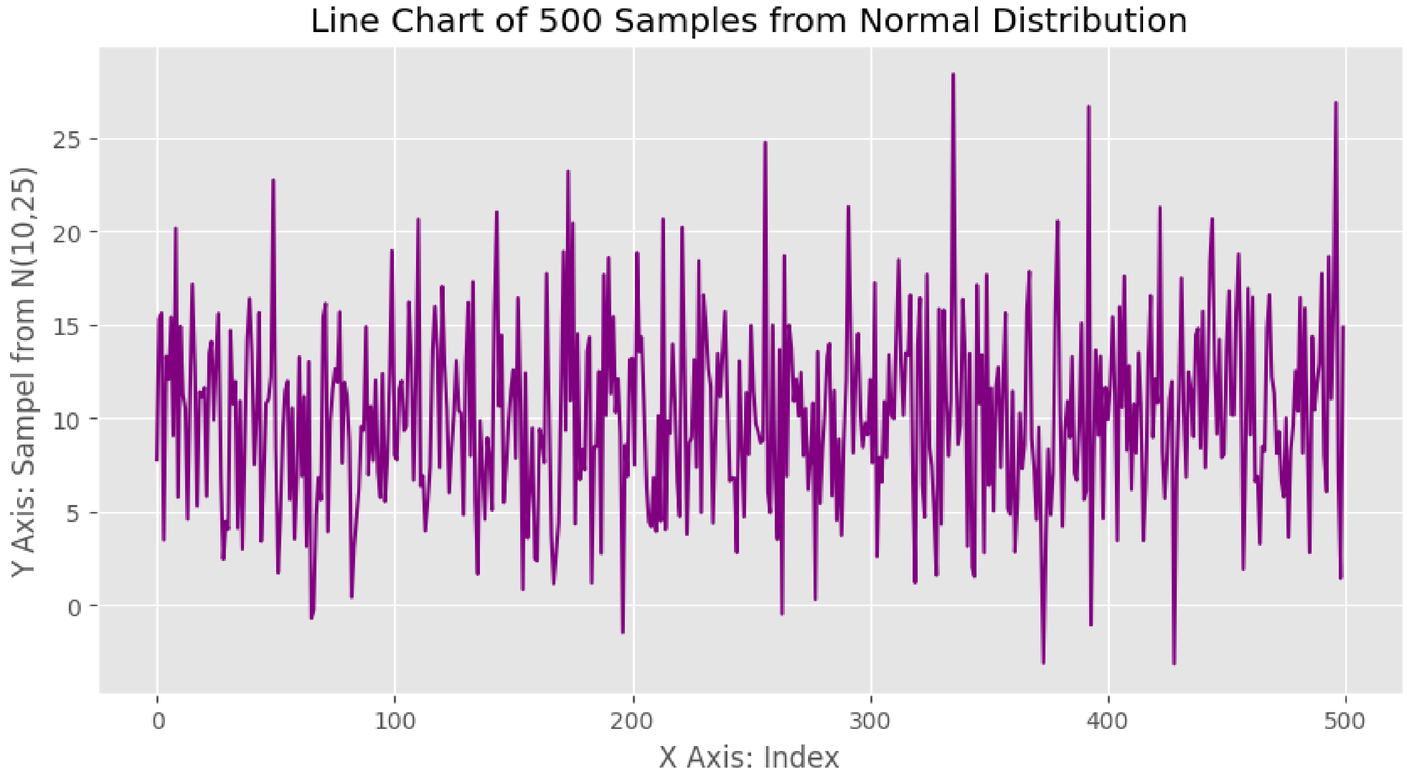
- ❑ Supports multiple chart types – Line plots, bar charts, histograms, scatter plots, and more
- ❑ Highly customizable – Modify colors, labels, legends, and styles
- ❑ Works with NumPy & SciPy – Ideal for scientific computing and data analysis
- ❑ Integrates with Pandas & Seaborn – Enhances data visualization in Python
- ❑ Supports interactive plotting – Works in Jupyter Notebook and Google Colab

Line Plot

```
pjme = np.random.normal(10, 5, 500)
plt.figure(figsize=(10, 5))
plt.plot(pjme, color="purple")
plt.xlabel("X Axis: Index")
plt.ylabel("Y Axis: Sampel from N(10,25)")
plt.title("Line Chart of 500 Samples from Normal Distribution")
plt.show()
```

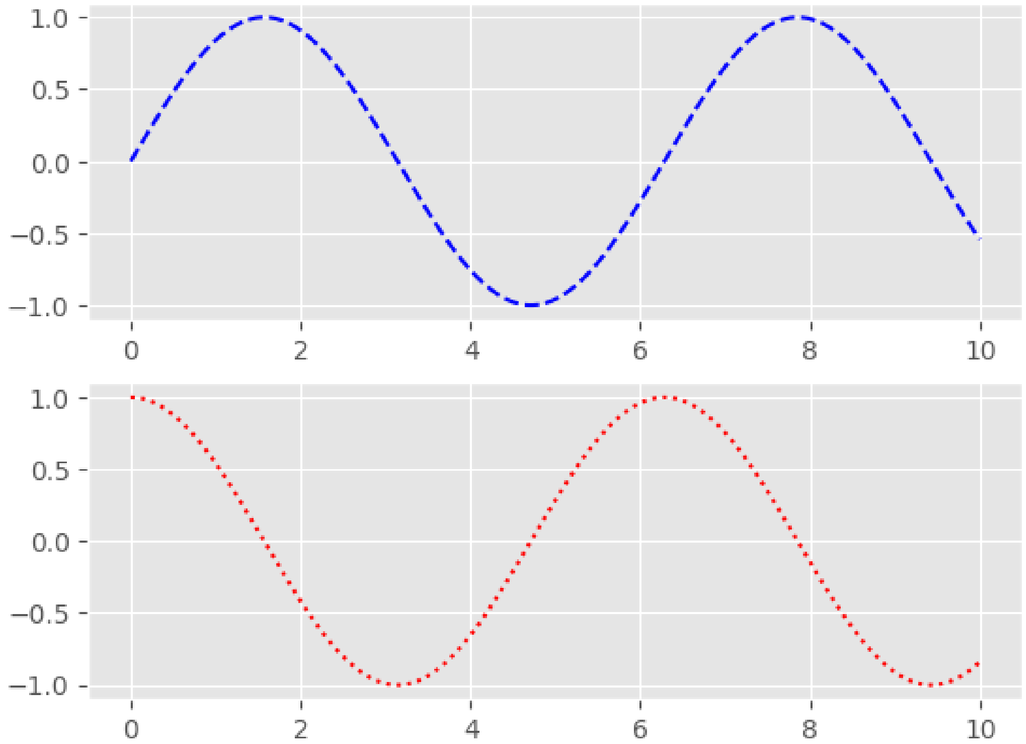
Add Label for x and y axis

Display the plot



Extra: Color and Styles

```
# First create a grid of plots
# ax will be an array of two Axes objects
fig, ax = plt.subplots(2)
# Call plot() method on the appropriate object
x = np.linspace(0, 10, 1000)
ax[0].plot(x, np.sin(x), linestyle='--', color='blue')
ax[1].plot(x, np.cos(x), linestyle=':', color='red')
```

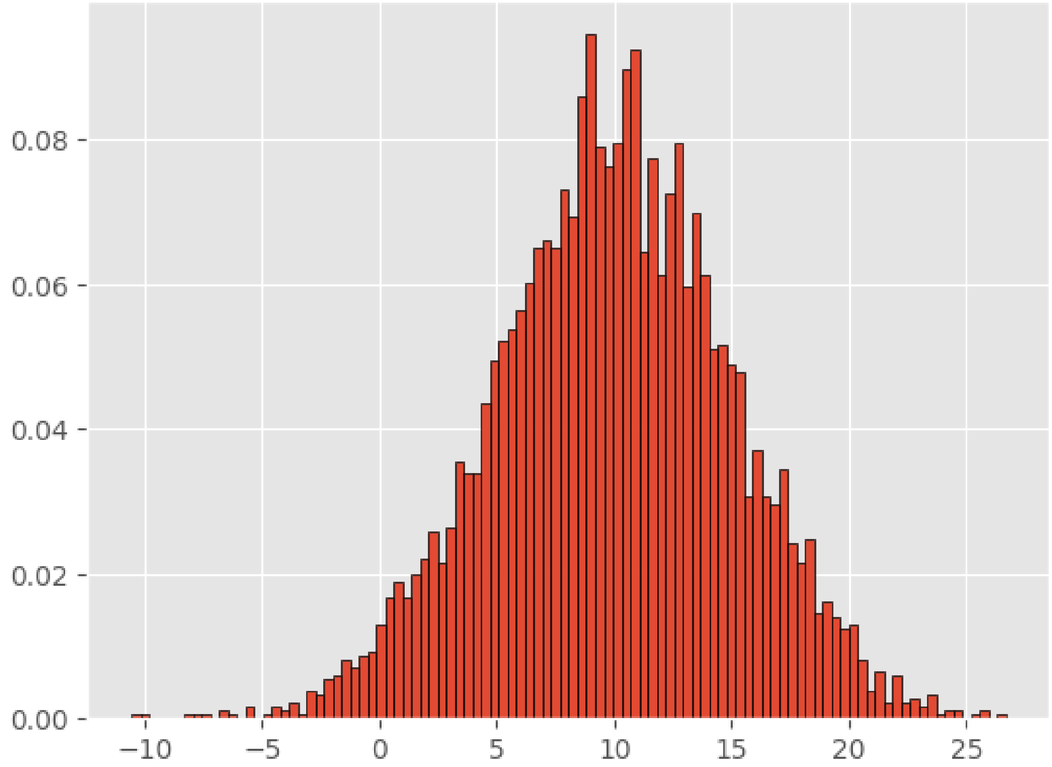


Histogram

```
pjme = np.random.normal(10, 5, 5000)
plt.hist(pjme, edgecolor="black", density=True, bins=50)
# Save as PNG
plt.savefig("histogram.png", dpi=300, bbox_inches="tight")
# Display the plot
plt.show()
```

density = True means it shows the relative frequency

Save the plot

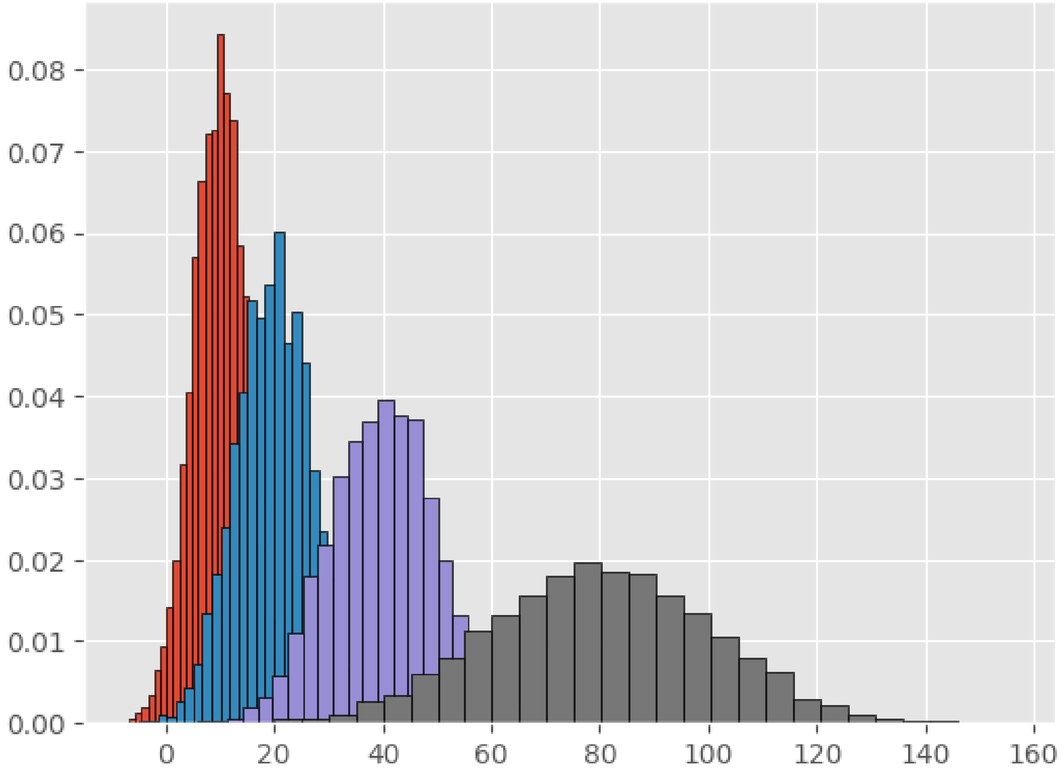


Histogram

Compare multiple distributions with different mean and standard deviation

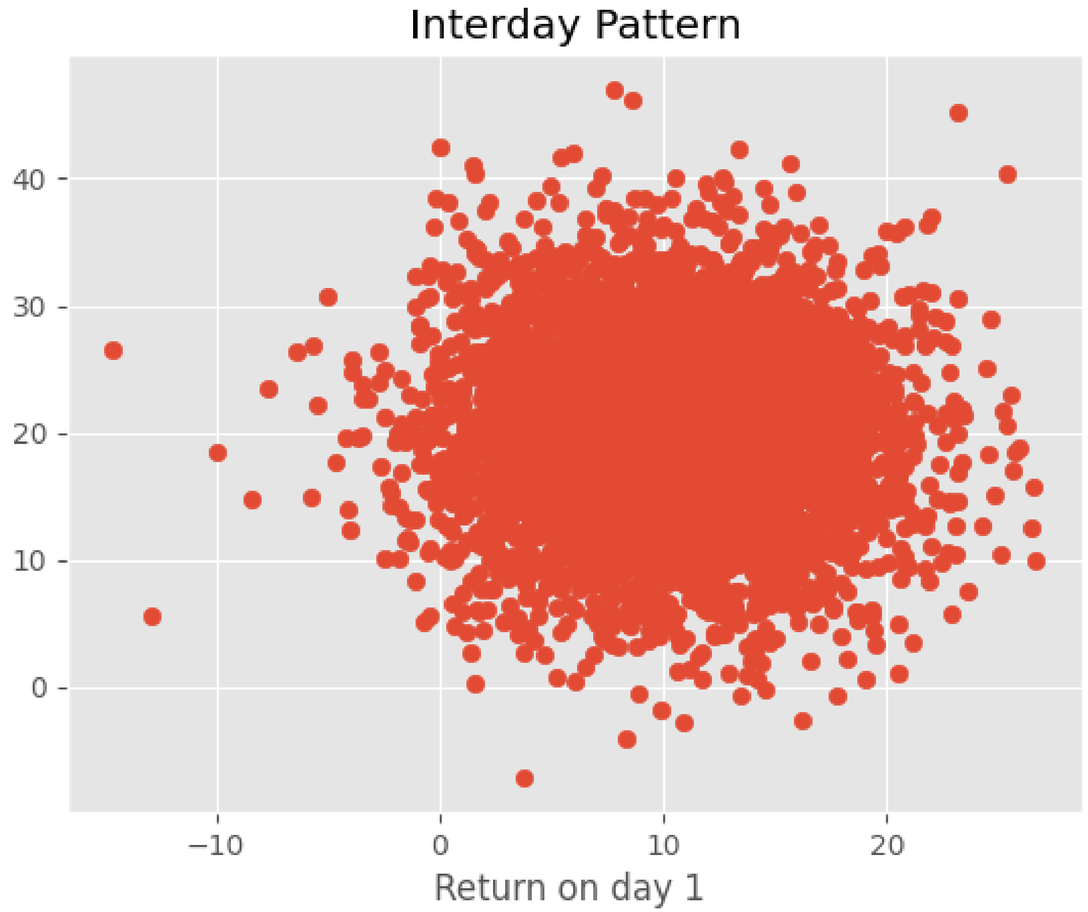
```
pjme1 = np.random.normal(10,5,5000)
pjme2 = np.random.normal(20,7,5000)
pjme3 = np.random.normal(40,10,5000)
pjme4 = np.random.normal(80,20,5000)
```

```
plt.hist(pjme1,bins=30,density=True, edgecolor="black" )
plt.hist(pjme2,bins=30,density=True, edgecolor="black" )
plt.hist(pjme3,bins=30,density=True, edgecolor="black" )
plt.hist(pjme4,bins=30,density=True, edgecolor="black" )
plt.show()
```



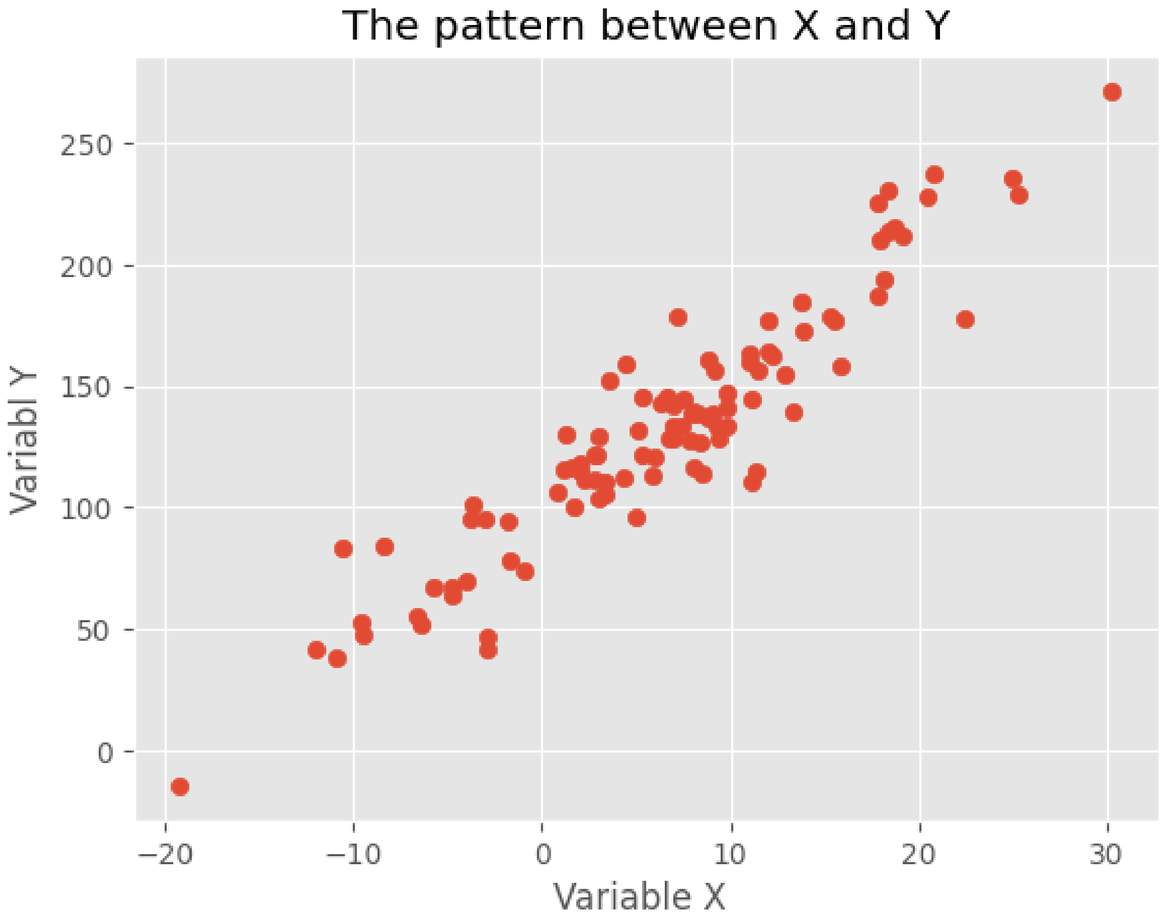
Scatterplot

```
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
ax1.scatter(pjme1, pjme2)
ax1.set_xlabel("Return on day 1 ")
ax1.set_ylabel("Return on day 2")
ax1.set_title("Interday Pattern")
```



Scatterplot

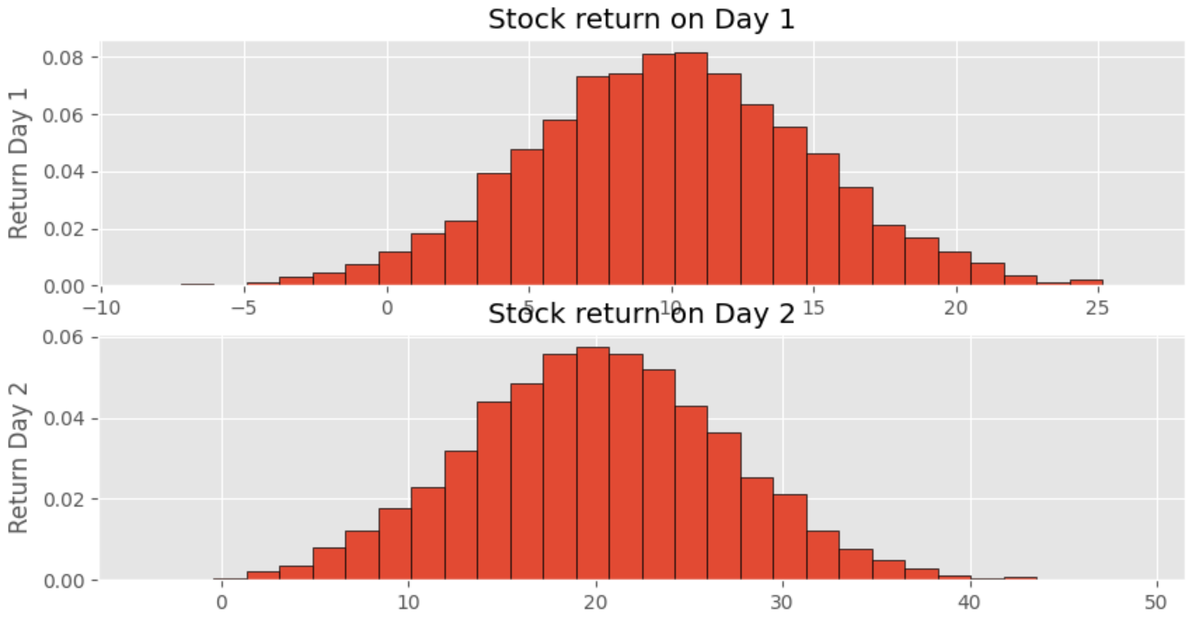
The example in Notebook is to generate data (X,Y) with the following relationship Y=100+5X+ϵ



Extra: Figure is an Object

In Matplotlib, **figure** is an **object** that serves as the **container** for all elements of a plot, such as axes, labels, legends, and more

```
# Create a figure object
fig = plt.figure(figsize=(10, 5))
# A Figure can contain multiple Axes (subplots)
# Figure (fig) is the container.
# Axes (axes1, axes2) are individual plots inside the figure
axes1 = fig.add_subplot(2, 1, 1)
axes1.hist(pjme1, bins=30, density=True, edgecolor="black" )
axes1.set_title("Stock return on Day 1")
axes1.set_ylabel("Return Day 1")
axes2 = fig.add_subplot(2, 1, 2)
axes2.hist(pjme2, bins=30, density=True, edgecolor="black" )
axes2.set_title("Stock return on Day 2")
axes2.set_ylabel("Return Day 2")
fig.show()
```

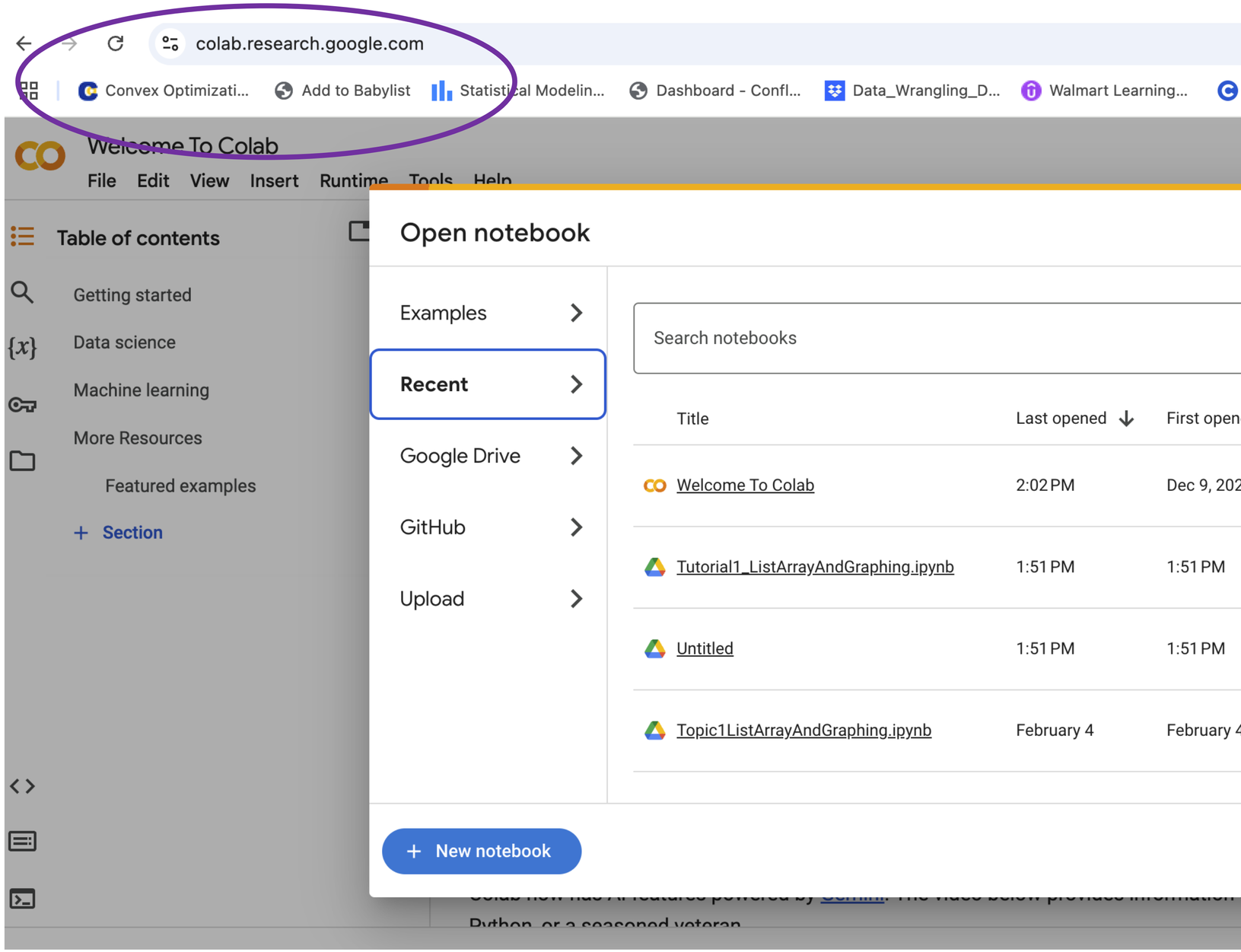


In-Class Exercise 4

1. We will use a **random number generator** to **simulate** daily stock **price changes** over **252 days**. For simplicity, we assume that these daily changes follow a **standard normal distribution**
2. Using NumPy's **cumulative sum** method, we will compute the accumulated sum of daily changes, which serves as a proxy for stock price movements
3. Finally, we will **plot** the simulated stock price to visualize its trend

Appendix: Launch Colab

- ❑ You can create a new Notebook
- ❑ Or open an existing Notebook in the folder
- ❑ Or upload a Notebook from local



Summary 1: libraries we use often

Packages	Main Functionalities
NumPy	NumPy is a package for scientific computing. We mainly use it to generate random variables.
SciPy	SciPy provides algorithms for optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics, and many other classes of problems. We mainly use it to compute the cumulative distribution function.
Matplotlib	Matplotlib is a comprehensive library for creating **static, animated, and interactive visualizations** in Python
Pandas – "Excel of Python"	Pandas offers **data structures and operations** for manipulating **numerical tables and time series**. This library is our main focus.

Summary 2: Common Data Types

Categories	Types	Description	Examples
Numerical	int	Integer (whole numbers)	x = 10
	float	Floating-point (decimal numbers)	y = 3.14
Sequence	str	String (text)	"Hello, Python!"
	list	Ordered, mutable collection	[1, "hello", 3.14, True]
Boolean	bool	Used for logical conditions	is_python_fun = True is_raining = False

Summary 3: Random Number Generator np.random

Distribution	Function	Examples
Uniform[0,1]	np.random.rand()	np.random.rand(5)
Standard Normal	np.random.randn()	np.random.randn(5)
Normal with mean and std	np.random.normal(mean, std, size)	np.random.normal(0, 1, 5) np.random.normal(10, 20, (2, 3))

Summary 4: Indexing and Slicing in Array

NumPy arrays support **indexing** (accessing elements) and **slicing** (extracting subarrays) to efficiently manipulate data

- ❑ a and b are index
- ❑ start: starting index (inclusive)
- ❑ stop: ending index (exclusive)

Operation	**Example**
Indexing 1D	arr[a]
Slicing 1D	arr[start: stop]
Indexing 2D	matrix[a, b]
Slicing 2D	matrix[a:, start: stop]

Summary 5: SciPy Statistical Methods

Operation	Function
Find Probability in a Normal Distribution	scipy.stats.norm.cdf(x, loc=0, scale=1)
Find Critical Value in a Normal Distribution	scipy.stats.norm.ppf(q, loc=0, scale=1)

Take away from Topic 1

- ❑ The common data types: int, float, bool and str
- ❑ List: indexing and slicing, list comprehension
- ❑ Array: indexing and slicing, element-wise operation, random number generator, aggregation methods (mean, sum…)
- ❑ SciPy: calculate probability and find critical value
- ❑ Plotting: line plot, histogram, scatter plot