Machine Learning Lab - preliminaries

Dr. Diptadip Maiti
Associate Professor, Department of CSE,
MCKV Institute of Engineering, 243, G T Road North, Liluah, Howrah, West Bengal 711204
+91-7003173682, +91-9433736910, diptadipmaiti@gmail.com

Contents

1 Introduction to Python

Python is a popular, high-level, interpreted programming language known for its readability, simplicity, and vast ecosystem. Developed by Guido van Rossum and first released in 1991, Python emphasizes code readability with its significant use of indentation.

1.1 Applications of Python

Python is used in various domains including:

- Web development (using frameworks like Django, Flask)
- Data science and analytics
- Machine learning and AI
- Scientific computing

- Automation and scripting
- Game development
- Desktop and GUI applications

1.2 Key Features

- Easy to learn and syntax resembles English
- Interpreted language - code is executed line by line
- Dynamically typed - no need to declare variable types
- Extensive standard library
- Supports multiple programming paradigms (procedural, OOP, functional)
- Large and active community

1.3 Installation and Setup

1.3.1 Downloading Python

Visit the official Python website at https://www.python.org and download the installer suitable for your operating system.

1.3.2 Installing Python

Run the installer. Ensure that the checkbox "Add Python to PATH" is selected. Complete the installation process by following the prompts.

1.3.3 Choosing an IDE

You can use any text editor or IDE. Popular choices include:

- Visual Studio Code
- PyCharm
- Jupyter Notebook (great for data science)
- Google Colab (online Jupyter environment)

1.3.4 Verify Installation

Open a terminal or command prompt and type:

```
python --version
```

1.4 Basic Syntax

1.5 Writing Python Code

Python uses indentation instead of braces to define code blocks. This makes the code more readable and less cluttered.

Hello World Program

```
print("Hello, World!")
```

This prints "Hello, World!" to the console.

1.5.1 Variables and Data Types

Variables in Python do not require explicit declaration. The type is inferred at runtime.

Examples

```
name = "Alice"        # str
age = 25              # int
height = 5.8          # float
is_student = True     # bool
```

1.5.2 Common Data Structures

- **List:** Ordered, mutable collection.
- **Tuple:** Ordered, immutable collection.
- **Set:** Unordered collection of unique elements.
- **Dictionary:** Key-value pairs.

Examples of List and Dictionary

```
fruits = ['apple', 'banana', 'cherry']
person = {'name': 'Bob', 'age': 30}
```

1.6 Control Flow Statements

1.6.1 Conditional Statements

```
if age >= 18:
    print("Adult")
elif age >= 13:
    print("Teen")
else:
    print("Child")
```

1.6.2 Loops

For Loop

```
for i in range(5):
    print(i)
```

While Loop

```
i = 0
while i < 5:
    print(i)
    i += 1
```

1.7 Functions

Functions are used to encapsulate reusable logic.

1.7.1 Defining a Function

```
def greet(name):
    return "Hello, " + name

print(greet("Alice"))
```

1.7.2 Default and Keyword Arguments

```
def greet(name="Guest"):
    print("Hello,", name)

greet()
greet("Bob")
```

1.8 Modules and Libraries

Modules help organize and reuse code. You can import built-in or third-party libraries.

Example: Using math module

```
import math

print(math.sqrt(16))
```

1.8.1 Popular Python Libraries

- `numpy`: Numerical computing and arrays
- `pandas`: Data analysis and manipulation
- `matplotlib`: Visualization and plotting
- `requests`: Making HTTP requests

1.9 File Handling

Reading and Writing Files Python provides functions to work with files.

Writing to a File

```
with open("output.txt", "w") as f:
    f.write("Hello, file!")
```

Reading from a File

```
with open("output.txt", "r") as f:
    content = f.read()
    print(content)
```

1.10 Exception Handling

Use try-except blocks to handle exceptions and prevent program crashes.

Example

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
```

1.11 Object-Oriented Programming

1.11.1 Classes and Objects

Python supports object-oriented programming to structure code.

1.11.2 Defining a Class

```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def details(self):
        print(f"{self.brand} {self.model}")

my_car = Car("Toyota", "Corolla")
my_car.details()
```

1.11.3 Inheritance

```
class ElectricCar(Car):
    def battery(self):
        print("Battery powered")

ecar = ElectricCar("Tesla", "Model 3")
ecar.details()
ecar.battery()
```

Python's simple syntax and powerful features make it an excellent choice for both beginners and experts. It is suitable for a wide range of applications, and its ecosystem is rich and growing.

Further Reading

- Official Documentation: https://docs.python.org/3/
- Real Python: https://realpython.com
- W3Schools Python: https://www.w3schools.com/python/

2 Introduction to Numpy

NumPy (Numerical Python) is a powerful library for numerical computing in Python. It supports large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions to operate on these arrays.

Key Features:

- Efficient array computation
- Broadcasting capabilities
- Tools for integrating C/C++
- Useful linear algebra, Fourier transform, and random number capabilities

2.1 Installation

To install NumPy, use pip:

```
pip install numpy
```

You can also use it via Jupyter or Google Colab where it is usually pre-installed.

2.2 Importing NumPy

```
import numpy as np
```

2.3 Creating Arrays

2.3.1 From Python Lists

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

2.3.2 Multi-dimensional Arrays

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

2.3.3 Predefined Arrays

```
np.zeros((2, 3))        # 2x3 array of zeros
np.ones((3, 3))         # 3x3 array of ones
np.eye(3)               # 3x3 Identity matrix
np.arange(0, 10, 2)     # array([0, 2, 4, 6, 8])
np.linspace(0, 1, 5)    # 5 values from 0 to 1
```

2.4 Array Attributes

```
a = np.array([[1,2,3],[4,5,6]])
a.shape         # (2, 3)
a.ndim          # 2 (dimensions)
a.dtype         # int64 (data type)
a.size          # 6 (total elements)
a.itemsize      # bytes per element
```

2.5 Indexing and Slicing

2.5.1 1D Arrays

```
a = np.array([10, 20, 30, 40])
a[1]         # 20
a[1:3]       # array([20, 30])
a[-1]        # 40
```

2.5.2 2D Arrays

```
b = np.array([[1,2,3], [4,5,6]])
b[1, 2]      # 6
b[:, 1]      # array([2, 5])
```

2.6 Array Operations

2.6.1 Arithmetic Operations

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
a + b        # array([5, 7, 9])
a * b        # element-wise multiplication
a ** 2       # square each element
```

2.6.2 Matrix Multiplication

```
np.dot(a, b)  # Dot product
```

2.7 Broadcasting

Broadcasting allows NumPy to work with arrays of different shapes.

```
a = np.array([1, 2, 3])
b = 2
a + b         # array([3, 4, 5])
```

2.8 Universal Functions (ufuncs)

```
np.sqrt([1, 4, 9])           # array([1., 2., 3.])
np.exp([0, 1, 2])            # e^x
np.log([1, np.e, np.e**2]) # natural log
```

2.9 Aggregate Functions

```
a = np.array([[1,2,3], [4,5,6]])
a.sum()              # 21
a.mean()             # 3.5
a.std()              # standard deviation
a.max(), a.min()
a.sum(axis=0)        # column-wise
a.sum(axis=1)        # row-wise
```

2.10 Reshaping and Stacking

2.10.1 Reshape

```
a = np.arange(12)
a.reshape(3, 4)
```

2.10.2 Stacking

```
a = np.array([1, 2])
b = np.array([3, 4])
np.vstack((a, b))    # vertical stack
np.hstack((a, b))    # horizontal stack
```

2.11 Random Numbers

```
np.random.rand(2,3)      # Uniform [0,1)
np.random.randn(3)       # Standard normal
np.random.randint(1, 10, size=(2,2))  # Random ints
```

2.12 Practical Example: Basic Statistics

```
data = np.random.randn(1000)
print("Mean:", np.mean(data))
print("Std Dev:", np.std(data))
print("Max:", np.max(data))
print("Min:", np.min(data))
```

NumPy is essential for efficient numerical computations in Python. It provides powerful tools for working with large arrays and matrices, performing vectorized operations, and integrating with other scientific libraries.

Further Reading

- Official Docs: https://numpy.org/doc/
- NumPy Quickstart: https://numpy.org/doc/stable/user/quickstart.html
- Book: *Python for Data Analysis* by Wes McKinney

3 Introduction to SciPy

SciPy (Scientific Python) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. Built on top of NumPy, it provides high-level functions for:

- Numerical integration and differentiation
- Optimization and root finding
- Linear algebra and statistics
- Signal and image processing

3.1 Installation

```
pip install scipy
```

3.2 SciPy Submodules

Commonly used submodules include:

- `scipy.integrate` – Integration and ODE solvers
- `scipy.optimize` – Optimization algorithms
- `scipy.linalg` – Linear algebra
- `scipy.stats` – Probability distributions and statistics
- `scipy.signal` – Signal processing

- `scipy.fft` – Fast Fourier Transforms
- `scipy.interpolate` – Interpolation
- `scipy.special` – Special mathematical functions

3.3 Integration and Differentiation

```
from scipy.integrate import quad

result, error = quad(lambda x: x**2, 0, 3)
print("Integral:", result)
```

3.3.1 Solving ODEs

```
from scipy.integrate import solve_ivp

def f(t, y): return -2*y
sol = solve_ivp(f, [0, 5], [1])
```

3.4 Optimization

```
from scipy.optimize import minimize

f = lambda x: x**2 + 10*np.sin(x)
res = minimize(f, x0=0)
print("Minimum at:", res.x)
```

3.4.1 Root Finding

```
from scipy.optimize import root

g = lambda x: x**3 - 1
sol = root(g, x0=1)
print("Root:", sol.x)
```

3.5 Linear Algebra

```
from scipy import linalg
import numpy as np

A = np.array([[3, 2], [1, 4]])
b = np.array([8, 7])

x = linalg.solve(A, b)
print("Solution:", x)
```

3.5.1 Eigenvalues and Eigenvectors

```
vals, vecs = linalg.eig(A)
print("Eigenvalues:", vals)
```

3.6 Statistics

```
from scipy import stats

data = np.random.normal(loc=0, scale=1, size=1000)
print("Mean:", np.mean(data))
print("Kurtosis:", stats.kurtosis(data))
print("Skewness:", stats.skew(data))
```

3.6.1 PDF and CDF

```
x = np.linspace(-3, 3, 100)
pdf = stats.norm.pdf(x)
cdf = stats.norm.cdf(x)
```

3.7 Signal Processing

```
from scipy.signal import butter, lfilter

b, a = butter(3, 0.05)
filtered = lfilter(b, a, np.random.rand(100))
```

3.8 Special Functions

```
from scipy import special

print("Gamma(5):", special.gamma(5))
print("Bessel J0(2):", special.j0(2))
```

3.9 Interpolation

```
from scipy.interpolate import interp1d

x = np.array([0, 1, 2])
y = np.array([0, 2, 3])
f = interp1d(x, y)
print("Interpolated:", f(1.5))
```

3.10 Fast Fourier Transform (FFT)

```
from scipy.fft import fft, fftfreq

x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
yf = fft(y)
```

3.11 Image Processing (Optional)

```
from scipy import ndimage

image = np.random.rand(5, 5)
blurred = ndimage.gaussian_filter(image, sigma=1)
```

SciPy extends NumPy and provides a powerful collection of algorithms for scientific computing. It is essential for numerical simulations, signal processing, and applied mathematics in Python.

Further Resources

- Official Docs: https://docs.scipy.org/doc/scipy/
- SciPy Cookbook: https://scipy-cookbook.readthedocs.io/
- GitHub: https://github.com/scipy/scipy
- Book: *Python for Data Analysis* by Wes McKinney

4 Introduction to Pandas

pandas is a fast, powerful, flexible and easy-to-use open source data analysis and manipulation tool built on top of the Python programming language.

Key Features:

- Data structures: Series (1D) and DataFrame (2D)
- Integrated handling of missing data
- Label-based slicing, fancy indexing, and subsetting
- Powerful group-by functionality
- High-performance merge and join operations

4.1 Installation

Install pandas using pip:

```
pip install pandas
```

Import it using:

```
import pandas as pd
```

4.2 pandas Data Structures

4.2.1 Series

A one-dimensional labeled array.

```
import pandas as pd

s = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
print(s)
```

4.2.2 DataFrame

A two-dimensional labeled data structure with columns of potentially different types.

```
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['Delhi', 'Mumbai', 'Chennai']
}
df = pd.DataFrame(data)
print(df)
```

4.3 Indexing and Selection

4.3.1 Accessing Columns and Rows

```
df['Name']              # column access
df.loc[0]               # row by label
df.iloc[1]              # row by index
df[0:2]                 # slice rows
```

4.3.2 Boolean Indexing

```
df[df['Age'] > 28]
```

4.4 Common DataFrame Operations

```
df.head()               # first 5 rows
df.tail()               # last 5 rows
df.shape                # (rows, columns)
df.columns              # column labels
df.describe()           # summary statistics
df.info()               # column info
```

4.4.1 Modifying Data

```
df['Salary'] = [50000, 60000, 70000]
df.at[0, 'Age'] = 26
```

4.5 Handling Missing Data

```
df.isnull()             # detect missing
df.dropna()             # drop missing
df.fillna(0)            # fill missing with value
```

4.6 GroupBy Operations

Grouping and aggregating data.

```
df.groupby('City').mean()
```

Example

```
grouped = df.groupby('City')
grouped['Salary'].sum()
```

4.7 Merging, Joining, Concatenating

4.7.1 Concatenation

```
pd.concat([df1, df2])
```

4.7.2 Merging

```
pd.merge(df1, df2, on='ID')
```

4.7.3 Joining

```
df1.join(df2, how='inner')
```

4.8 Working with Files

4.8.1 Reading from Files

```
df = pd.read_csv('data.csv')
df = pd.read_excel('data.xlsx')
```

4.8.2 Writing to Files

```
df.to_csv('output.csv', index=False)
df.to_excel('output.xlsx', index=False)
```

4.9 Time Series

```
dates = pd.date_range('20230101', periods=6)
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('
    ABCD'))
```

4.10 Practical Example: Salary Analysis

```
data = {
    'Department': ['HR', 'IT', 'Finance', 'HR', 'IT'],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Salary': [50000, 60000, 70000, 55000, 62000]
}
df = pd.DataFrame(data)
avg_salary = df.groupby('Department')['Salary'].mean()
print(avg_salary)
```

4.11 Visualization Integration (optional)

Use pandas with matplotlib or seaborn for visual plots.

```
import matplotlib.pyplot as plt

df['Salary'].plot(kind='bar')
plt.title("Employee Salaries")
plt.show()
```

pandas is essential for real-world data analysis in Python. It simplifies tasks such as reading files, cleaning data, performing transformations, grouping, and time series analysis.

Further Reading

- Official Docs: https://pandas.pydata.org/docs/
- pandas Tutorials: https://pandas.pydata.org/docs/getting_started/index.html
- Book: *Python for Data Analysis* by Wes McKinney

5 Introduction to Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of formats and interactive environments. It is highly customizable and supports multiple backends.

Key Features:

- Support for line plots, bar charts, scatter plots, histograms, pie charts, etc.
- Integration with NumPy and pandas
- Fully customizable plots
- Object-oriented interface

5.1 Installation

Install using pip:

```
pip install matplotlib
```

Import with:

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

5.2 Basic Plotting

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y)
plt.show()
```

5.3 Adding Titles and Labels

```
plt.plot(x, y)
plt.title("Sample Line Plot")
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
plt.show()
```

5.4 Legends

```
plt.plot(x, y, label='Data 1')
plt.legend()
plt.show()
```

5.5 Multiple Plots in One Figure

```
y2 = [5, 15, 20, 25]
plt.plot(x, y, label='Line 1')
plt.plot(x, y2, label='Line 2', linestyle='--')
plt.legend()
plt.show()
```

5.6 Plot Types

5.6.1 Line Plot

```
plt.plot(x, y)
```

5.6.2 Bar Chart

```
plt.bar(x, y)
```

5.6.3 Histogram

```
import numpy as np
data = np.random.randn(1000)
plt.hist(data, bins=30)
plt.show()
```

5.6.4 Scatter Plot

```
plt.scatter(x, y)
```

5.6.5 Pie Chart

```
labels = ['A', 'B', 'C', 'D']
sizes = [15, 30, 45, 10]
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.axis('equal')
plt.show()
```

5.7 Customization

5.7.1 Color, Marker, Line Style

```
plt.plot(x, y, color='green', marker='o', linestyle='--')
```

5.7.2 Grid and Limits

```
plt.grid(True)
plt.xlim(0, 5)
plt.ylim(0, 35)
```

5.8 Subplots

5.8.1 Using `subplot()`

```
plt.subplot(2, 1, 1)
plt.plot(x, y)
plt.title("First Plot")

plt.subplot(2, 1, 2)
plt.plot(x, [v**2 for v in y])
plt.title("Second Plot")

plt.tight_layout()
plt.show()
```

5.8.2 Using subplots()

```
fig, axs = plt.subplots(1, 2)
axs[0].bar(x, y)
axs[1].scatter(x, y)
plt.tight_layout()
plt.show()
```

5.9 Saving Figures

```
plt.plot(x, y)
plt.savefig("my_plot.png")
```

5.10 Object-Oriented API

```
fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_title("OO Style")
ax.set_xlabel("X")
ax.set_ylabel("Y")
plt.show()
```

5.11 Integration with pandas

```
import pandas as pd

data = {
    'Month': ['Jan', 'Feb', 'Mar', 'Apr'],
    'Sales': [200, 300, 250, 400]
}
df = pd.DataFrame(data)
df.plot(x='Month', y='Sales', kind='bar')
plt.show()
```

5.12 Practical Example: Visualizing Sales Data

```
months = ['Jan', 'Feb', 'Mar', 'Apr']
sales_2023 = [300, 400, 350, 450]
sales_2024 = [320, 410, 370, 480]

plt.plot(months, sales_2023, label='2023')
plt.plot(months, sales_2024, label='2024', linestyle='--')
plt.title('Monthly Sales Comparison')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.legend()
plt.grid(True)
plt.show()
```

Matplotlib is a versatile and powerful tool for all kinds of visualizations. With both procedural and object-oriented interfaces, it's ideal for beginners and professionals working with data.

Further Reading

- Official Documentation: https://matplotlib.org/stable/contents.html
- Pyplot Tutorial: https://matplotlib.org/stable/tutorials/introductory/pyplot.html
- Book: *Python Data Science Handbook* by Jake VanderPlas

6 Introduction to Seaborn

Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. It works well with pandas DataFrames.

6.1 Installation

```
pip install seaborn
```

6.2 Import and Dataset Loading

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load built-in dataset
tips = sns.load_dataset("tips")
print(tips.head())
```

6.3 Themes and Styles

```
sns.set_style("whitegrid")  # Options: white, dark, whitegrid, darkgrid
    , ticks
```

6.4 Basic Plots

6.4.1 Distribution Plot

```
sns.histplot(data=tips, x="total_bill", kde=True)
plt.show()
```

6.4.2 Box Plot

```
sns.boxplot(x="day", y="total_bill", data=tips)
plt.show()
```

6.4.3 Violin Plot

```
sns.violinplot(x="day", y="total_bill", data=tips)
plt.show()
```

6.4.4 Strip Plot

```
sns.stripplot(x="day", y="total_bill", data=tips, jitter=True)
plt.show()
```

6.5 Categorical Plots

6.5.1 Bar Plot

```
sns.barplot(x="sex", y="total_bill", data=tips)
plt.show()
```

6.5.2 Count Plot

```
sns.countplot(x="day", data=tips)
plt.show()
```

6.6 Statistical Plots

6.6.1 Scatter Plot with Hue

```
sns.scatterplot(x="total_bill", y="tip", hue="smoker", data=tips)
plt.show()
```

6.6.2 Linear Regression Plot

```
sns.lmplot(x="total_bill", y="tip", data=tips)
```

6.7 Heatmaps and Correlation Matrix

```
corr = tips.corr()
sns.heatmap(corr, annot=True, cmap="coolwarm")
plt.show()
```

6.8 Pairplot

```
sns.pairplot(tips, hue="sex")
plt.show()
```

6.9 FacetGrid: Multiple Subplots

```
g = sns.FacetGrid(tips, col="sex", row="time")
g.map(sns.scatterplot, "total_bill", "tip")
plt.show()
```

6.10 Customizing Plots

```
sns.set_palette("pastel")
sns.boxplot(x="day", y="total_bill", data=tips)
plt.title("Box Plot of Total Bill by Day")
plt.xlabel("Day of Week")
plt.ylabel("Total Bill ($)")
plt.show()
```

6.11 Combining with pandas and matplotlib

```
import pandas as pd
avg_tip = tips.groupby("day")["tip"].mean().reset_index()
sns.barplot(x="day", y="tip", data=avg_tip)
plt.title("Average Tip by Day")
plt.show()
```

6.12 Real-World Example: Titanic Dataset

```
titanic = sns.load_dataset("titanic")
sns.countplot(x="class", hue="survived", data=titanic)
plt.title("Survival Count by Class")
plt.show()
```

Seaborn makes statistical visualization in Python simple and elegant. It is highly integrated with pandas and Matplotlib, and is ideal for exploratory data analysis (EDA).

Further Resources

- Official Docs: https://seaborn.pydata.org/
- Gallery: https://seaborn.pydata.org/examples/index.html
- Book: *Python for Data Analysis* by Wes McKinney

7 Introduction to scikit-learn

scikit-learn is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. Built on NumPy, SciPy, and Matplotlib, it's simple and efficient.

Features:

- Classification, Regression, Clustering

- Dimensionality Reduction
- Model Selection and Pipelines
- Preprocessing and Feature Engineering

7.1 Installation

Install via pip:

```
pip install scikit-learn
```

7.2 Supervised vs Unsupervised Learning

- **Supervised Learning**: Learns from labeled data. (e.g., Classification, Regression)
- **Unsupervised Learning**: Learns from data without labels. (e.g., Clustering, PCA)

7.3 Loading Datasets

scikit offers built-in datasets such as iris, digits, wine, and breast cancer.

```
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target
```

7.4 Data Preprocessing

7.4.1 Standardization

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

7.4.2 Label Encoding

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
y_encoded = le.fit_transform(y)
```

7.5 Splitting the Dataset

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

7.6 Classification Example: Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

model = LogisticRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

7.7 Classification: k-Nearest Neighbors

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
print("Test Accuracy:", knn.score(X_test, y_test))
```

7.8 Regression: Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

linreg = LinearRegression()
linreg.fit(X_train, y_train)
y_pred = linreg.predict(X_test)
print("MSE:", mean_squared_error(y_test, y_pred))
```

7.9 Regression: Decision Tree

```
from sklearn.tree import DecisionTreeRegressor

tree = DecisionTreeRegressor()
tree.fit(X_train, y_train)
print("Score:", tree.score(X_test, y_test))
```

7.10 Unsupervised Learning: K-Means Clustering

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X)
print("Cluster Centers:\n", kmeans.cluster_centers_)
print("Labels:", kmeans.labels_)
```

7.11 Model Evaluation

7.11.1 Cross-Validation

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5)
print("Cross-Validation Scores:", scores)
print("Average Score:", scores.mean())
```

7.12 Pipelines

```
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', LogisticRegression())
])

pipeline.fit(X_train, y_train)
print("Pipeline Accuracy:", pipeline.score(X_test, y_test))
```

7.13 Visualization Example with PCA

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

plt.scatter(X_reduced[:,0], X_reduced[:,1], c=y)
plt.title("PCA Projection of Iris")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

`scikit-learn` offers a consistent, simple API for building machine learning pipelines and models. It is a must-learn tool for any data scientist or ML practitioner.

Further Resources

- Official Site: https://scikit-learn.org/stable/
- User Guide: https://scikit-learn.org/stable/user_guide.html
- Course: https://scikit-learn.org/stable/tutorial/index.html
- Book: *Hands-On Machine Learning with Scikit-Learn* by Aurélien Géron

8 Introduction to Keras

Keras is a high-level deep learning API written in Python. It is user-friendly, modular, and runs on top of TensorFlow, making it ideal for quick prototyping and real-world deployment.

Advantages:

- Easy-to-use and intuitive API
- Built-in support for standard layers and optimizers
- Supports CNNs, RNNs, Autoencoders, and GANs
- Cross-platform (TensorFlow, JAX, PyTorch backends via Keras Core)

8.1 Installation

```
pip install tensorflow
# or
pip install keras
```

8.2 Typical Keras Workflow

1. Load and preprocess data
2. Define model architecture
3. Compile the model
4. Train the model
5. Evaluate performance
6. Predict or save the model

8.3 Loading Datasets

Keras provides utilities for popular datasets:

```
from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0
```

8.4 Building a Simple Neural Network

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
```

8.5 Model Compilation

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```


8.6 Model Training

```
model.fit(x_train, y_train, epochs=5, batch_size=32, validation_split
    =0.2)
```

8.7 Model Evaluation

```
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

8.8 Making Predictions

```
predictions = model.predict(x_test[:5])
predicted_classes = predictions.argmax(axis=1)
print(predicted_classes)
```

8.9 Saving and Loading a Model

```
# Save
model.save('my_model.h5')

# Load
from tensorflow.keras.models import load_model
model = load_model('my_model.h5')
```

8.10 Basic CNN for Image Classification

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
```

8.11 Keras Callbacks

Callbacks are used for monitoring training, early stopping, logging, etc.

```
from tensorflow.keras.callbacks import EarlyStopping

callback = EarlyStopping(monitor='val_loss', patience=3)
model.fit(x_train, y_train, epochs=20, validation_split=0.2, callbacks
    =[callback])
```

8.12 Practical Example: Binary Classification

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np

# Dummy data
X = np.random.rand(1000, 10)
y = np.random.randint(2, size=(1000, 1))

model = Sequential([
    Dense(64, activation='relu', input_shape=(10,)),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['
    accuracy'])
model.fit(X, y, epochs=10, batch_size=32)
```

Keras simplifies the process of building deep learning models. It's powerful enough for research and easy enough for beginners. For complex tasks, use the Functional API or subclassing models.

Further Resources

- Official Docs: https://keras.io/
- Keras Code Examples: https://keras.io/examples/
- TensorFlow + Keras Tutorials: https://www.tensorflow.org/tutorials/
- Book: *Deep Learning with Python* by François Chollet

9 Introduction to TensorFlow

TensorFlow is an end-to-end open-source platform for machine learning developed by Google. It supports both high-level APIs (like `tf.keras`) and low-level building blocks for advanced users.

Key Features:

- Automatic differentiation and gradient computation
- Scalable training on CPU, GPU, and TPU
- Supports production deployment via TensorFlow Serving, TFLite, and TF.js
- Integrates tightly with Keras

9.1 Installation

```
pip install tensorflow
```

9.2 Tensor Basics

```
import tensorflow as tf

# Create constant tensor
a = tf.constant([[1, 2], [3, 4]])
print(a)

# Create variable tensor
b = tf.Variable(tf.ones([2, 2]))
```

9.2.1 Tensor Properties

```
print(a.shape)
print(a.dtype)
print(tf.rank(a))
```

9.3 Eager Execution

`TensorFlow 2.x` runs operations eagerly by default.

```
x = tf.constant([1.0, 2.0, 3.0])
print(x + 1)
```

9.4 Automatic Differentiation

```
x = tf.Variable(3.0)

with tf.GradientTape() as tape:
    y = x**2

dy_dx = tape.gradient(y, x)
print(dy_dx)  # Output: 6.0
```

9.5 Using `tf.keras` for Neural Networks

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(10,)),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

9.6 Training the Model

```
import numpy as np

X = np.random.rand(1000, 10)
y = np.random.rand(1000, 1)

model.fit(X, y, epochs=5, batch_size=32)
```

9.7 Evaluation and Prediction

```
model.evaluate(X, y)
predictions = model.predict(X[:5])
```

9.8 Custom Training Loop (Low-Level API)

```
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)
loss_fn = tf.keras.losses.MeanSquaredError()

for epoch in range(5):
    with tf.GradientTape() as tape:
        predictions = model(X)
        loss = loss_fn(y, predictions)

    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables)
        )
```

9.9 Saving and Loading Models

```
# Save entire model
model.save('my_model')

# Load model
model = tf.keras.models.load_model('my_model')
```

9.10 Using TensorBoard

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir='./logs')
model.fit(X, y, epochs=5, callbacks=[tensorboard_callback])
```

9.11 Checking GPU Availability

```
print("Num GPUs:", len(tf.config.list_physical_devices('GPU')))
```

9.12 Practical Example: MNIST Digit Classifier

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0

x_train = x_train.reshape(-1, 28*28)
x_test = x_test.reshape(-1, 28*28)
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(784,)),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy',
   metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, validation_split=0.2)
model.evaluate(x_test, y_test)
```

TensorFlow provides the flexibility of low-level tensor manipulation and the simplicity of Keras for deep learning model building. It supports research and production-level deployments.

9.13 Further Resources

- Official Docs: https://www.tensorflow.org/
- TensorFlow Tutorials: https://www.tensorflow.org/tutorials

- Keras API Reference: https://www.tensorflow.org/api_docs/python/tf/keras
- Book: *Deep Learning with Python* by François Chollet

10 Introduction to PyTorch

PyTorch is an open-source deep learning framework developed by Facebook. It provides:

- Tensor computation (like NumPy) with GPU acceleration
- Dynamic computation graphs
- Autograd (automatic differentiation)
- Neural network APIs through `torch.nn`

10.1 Installation

Install PyTorch using pip:

```
pip install torch torchvision
```

10.2 Tensors in PyTorch

```
import torch

x = torch.tensor([1.0, 2.0, 3.0])
print(x.shape, x.dtype)

a = torch.ones((2, 3))
b = torch.zeros((2, 3))
c = a + b
```

10.2.1 Tensor Operations

```
mat1 = torch.randn(2, 3)
mat2 = torch.randn(3, 2)
result = torch.matmul(mat1, mat2)
```

10.3 Automatic Differentiation

```
x = torch.tensor(2.0, requires_grad=True)
y = x ** 2
y.backward()
print(x.grad)  # Output: 4.0
```

10.4 Neural Network using `torch.nn`

```
import torch.nn as nn

model = nn.Sequential(
    nn.Linear(10, 64),
    nn.ReLU(),
    nn.Linear(64, 1)
)
```

10.5 Loss Functions and Optimizers

```
loss_fn = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

10.6 Training a Model

```
X = torch.randn(100, 10)
y = torch.randn(100, 1)

for epoch in range(100):
    pred = model(X)
    loss = loss_fn(pred, y)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if epoch % 10 == 0:
        print(f"Epoch {epoch}, Loss: {loss.item()}")
```

10.7 Evaluation and Prediction

```
model.eval()
with torch.no_grad():
    sample = torch.randn(1, 10)
    prediction = model(sample)
    print(prediction)
```

10.8 Using GPU

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = model.to(device)
X = X.to(device)
y = y.to(device)
```

10.9 Saving and Loading Models

10.9.1 Saving

```
torch.save(model.state_dict(), 'model.pth')
```

10.9.2 Loading

```
model.load_state_dict(torch.load('model.pth'))
model.eval()
```

10.10 Practical Example: MNIST Classifier

```
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# Data loading
transform = transforms.ToTensor()
train_data = datasets.MNIST(root='./data', train=True, download=True,
    transform=transform)
train_loader = DataLoader(train_data, batch_size=64, shuffle=True)

# Define simple model
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        return self.linear_relu_stack(x)

model = MLP()
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Training loop
for epoch in range(5):
    for images, labels in train_loader:
        pred = model(images)
        loss = loss_fn(pred, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f"Epoch {epoch}, Loss: {loss.item()}")
```

PyTorch is powerful and flexible for building deep learning models, from quick prototypes to large-scale systems. Its dynamic computation graph is intuitive for debugging and experimentation.

10.11 Further Resources

- Official Website: https://pytorch.org/
- Tutorials: https://pytorch.org/tutorials/
- Documentation: https://pytorch.org/docs/stable/index.html
- Book: *Deep Learning with PyTorch* by Eli Stevens et al.

11 Introduction to OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It provides a common infrastructure for computer vision applications and accelerates the use of machine perception in commercial products.

11.1 Installation

```
pip install opencv-python
pip install opencv-python-headless  # if GUI is not required
```

11.2 Reading and Writing Images

```
import cv2

# Read an image
img = cv2.imread('image.jpg')

# Display image
cv2.imshow('Image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Save image
cv2.imwrite('output.jpg', img)
```

11.3 Image Properties and Operations

```
print(img.shape)  # Height, Width, Channels
resized = cv2.resize(img, (300, 300))
cropped = img[100:300, 200:400]
```

11.4 Drawing Shapes and Text

```
cv2.line(img, (50, 50), (200, 50), (255, 0, 0), 3)
cv2.rectangle(img, (100, 100), (200, 200), (0, 255, 0), 2)
cv2.circle(img, (150, 150), 40, (0, 0, 255), -1)
cv2.putText(img, 'OpenCV', (10, 300), cv2.FONT_HERSHEY_SIMPLEX, 1,
   (255,255,255), 2)
```

11.5 Color Spaces

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

11.6 Thresholding and Edge Detection

```
# Simple thresholding
_, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

# Canny edge detection
edges = cv2.Canny(img, 100, 200)
```

11.7 Blurring and Smoothing

```
blur = cv2.GaussianBlur(img, (5, 5), 0)
median = cv2.medianBlur(img, 5)
```

11.8 Morphological Operations

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
dilated = cv2.dilate(thresh, kernel, iterations=1)
eroded = cv2.erode(thresh, kernel, iterations=1)
```

11.9 Contour Detection

```
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.
   CHAIN_APPROX_SIMPLE)
cv2.drawContours(img, contours, -1, (0, 255, 0), 2)
```

11.10 Video Capture from Webcam

```
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    cv2.imshow("Webcam", frame)
    if cv2.waitKey(1) == ord('q'):
        break
```

```
cap.release()
cv2.destroyAllWindows()
```

11.11 Real-Time Face Detection

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + '
    haarcascade_frontalface_default.xml')

cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x,y,w,h) in faces:
        cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)

    cv2.imshow('Face Detection', frame)
    if cv2.waitKey(1) == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

OpenCV enables fast and efficient image processing, from basic operations to advanced computer vision tasks. It's widely used in industry and academia for face recognition, gesture control, object detection, and more.

11.12 Resources

- Official Site: https://opencv.org/
- Docs: https://docs.opencv.org/
- OpenCV GitHub: https://github.com/opencv/opencv
- Book: *Learning OpenCV* by Gary Bradski

12 Viva Question

[1 Mark] What is Python and why is it so popular?

Answer: Python is a high-level, interpreted programming language known for its simplicity and readability. It is popular due to its vast ecosystem, ease of use, and applicability across domains like web development, data science, and automation.

[1 Mark] How do you install Python on your system?

Answer: You can install Python by downloading the installer from https://www.python.org, running it, and ensuring that "Add Python to PATH" is selected during installation.

[1 Mark] What is the significance of indentation in Python?

Answer: In Python, indentation defines the structure of code blocks. Unlike other languages that use braces, Python uses whitespace to delimit code scopes.

[1 Mark] What are the basic data types in Python?

Answer: Basic data types in Python include `int`, `float`, `str`, and `bool`. Python is dynamically typed, so variable types are determined at runtime.

[1 Mark] What is the difference between a list and a tuple in Python?

Answer: A list is mutable, meaning its contents can be changed. A tuple is immutable and cannot be modified after creation.

[1 Mark] How do you write a loop in Python?

Answer: You can use `for` or `while` loops. For example: `for i in range(5): print(i)` iterates from 0 to 4.

[1 Mark] How are functions defined and called in Python?

Answer: Functions are defined using the `def` keyword. Example: `def greet(): return "Hello"` is called using `greet()`.

[1 Mark] What is a Python module and how is it used?

Answer: A module is a Python file containing code (functions, classes) that can be imported and reused. Example: `import math` allows access to mathematical functions.

[1 Mark] How does Python handle exceptions?

Answer: Python uses `try-except` blocks to handle exceptions. Example: `try: x=1/0 except ZeroDivisionError: print("Error")`.

[1 Mark] What is object-oriented programming in Python?

Answer: Python supports OOP, allowing users to define classes and create objects. It supports principles like inheritance, encapsulation, and polymorphism.

[1 Mark] What is NumPy and what is it used for?

Answer: NumPy is a Python library for numerical computing. It provides support for large multi-dimensional arrays and matrices, and a collection of mathematical functions to operate on them efficiently.

[1 Mark] How do you install NumPy?

Answer: You can install NumPy using pip with the command: `pip install numpy`.

[1 Mark] How do you import NumPy in your Python script?

Answer: NumPy is typically imported with the alias: `import numpy as np`.

[1 Mark] How do you create a NumPy array from a Python list?

Answer: Use `np.array(list)`. Example: `np.array([1, 2, 3])`.

[1 Mark] What is the difference between `np.zeros()` and `np.ones()`?

Answer: `np.zeros()` creates an array filled with zeros, while `np.ones()` creates an array filled with ones.

[1 Mark] What does `np.eye(3)` return?

Answer: It returns a 3x3 identity matrix.

[1 Mark] How is `np.arange()` different from `np.linspace()`?

Answer: `np.arange()` returns values with a specified step, whereas `np.linspace()` returns evenly spaced numbers over a specified interval.

[1 Mark] What is the use of `shape`, `ndim`, and `dtype` attributes in NumPy arrays?

Answer: `shape` gives the dimensions, `ndim` gives the number of dimensions, and `dtype` gives the data type of the array.

[1 Mark] How do you access elements in a 1D NumPy array?

Answer: Using indexing like `a[0]`, `a[-1]`, or slicing like `a[1:3]`.

[1 Mark] How do you access elements in a 2D array?

Answer: Using row and column indices like `a[1, 2]` or slicing like `a[:, 1]`.

[1 Mark] What is element-wise operation in NumPy?

Answer: Operations like addition, subtraction, multiplication done element-by-element on arrays of the same shape.

[1 Mark] What is the difference between element-wise multiplication and matrix multiplication in NumPy?

Answer: Element-wise uses `*`, matrix multiplication uses `np.dot()` or the `@` operator.

[1 Mark] What is broadcasting in NumPy?

Answer: Broadcasting is the ability of NumPy to perform operations on arrays of different shapes by automatically expanding them.

[1 Mark] Give an example of a universal function (ufunc) in NumPy.

Answer: `np.sqrt()`, `np.exp()`, and `np.log()` are common ufuncs.

[1 Mark] What is the purpose of aggregate functions like `sum()`, `mean()`, and `std()`?

Answer: They compute summary statistics like total, average, and standard deviation.

[1 Mark] How do you perform column-wise and row-wise operations using NumPy?

Answer: Use the `axis` parameter in functions like `sum(axis=0)` for columns and `sum(axis=1)` for rows.

[1 Mark] How do you reshape a NumPy array?

Answer: Use the `reshape()` method. Example: `a.reshape(3, 4)`.

[1 Mark] What is the difference between `vstack()` and `hstack()`?

Answer: `vstack()` stacks arrays vertically (row-wise), while `hstack()` stacks arrays horizontally (column-wise).

[1 Mark] How do you generate random numbers in NumPy?

Answer: Use `np.random.rand()`, `np.random.randn()`, or `np.random.randint()`.

[1 Mark] How can NumPy be useful in statistics?

Answer: NumPy provides functions to calculate mean, standard deviation, min, max, and more, making it suitable for statistical analysis.

[1 Mark] What is SciPy and what is its relationship with NumPy?

Answer: SciPy is a Python library built on top of NumPy that provides additional functionality for scientific and technical computing, including modules for optimization, integration, linear algebra, and statistics.

[1 Mark] How do you install the SciPy library?

Answer: You can install SciPy using pip: `pip install scipy`.

[1 Mark] What is the purpose of the `scipy.integrate` submodule?

Answer: It provides functions for numerical integration and solving ordinary differential

equations (ODEs).

[1 Mark] Which function is used to compute definite integrals in SciPy?

Answer: The function `quad()` from `scipy.integrate` is used for definite integration.

[1 Mark] How can you solve ODEs in SciPy?

Answer: You can use `solve_ivp()` from `scipy.integrate` to solve initial value problems for ODEs.

[1 Mark] What is the use of `scipy.optimize`?

Answer: It provides functions to minimize or maximize functions, and to find roots of equations.

[1 Mark] Which function finds the minimum of a scalar function?

Answer: `minimize()` from `scipy.optimize`.

[1 Mark] How do you find the root of a function in SciPy?

Answer: Use the `root()` function from `scipy.optimize`.

[1 Mark] Which submodule is used for linear algebra operations?

Answer: `scipy.linalg` is used for advanced linear algebra operations.

[1 Mark] How do you solve a system of linear equations in SciPy?

Answer: Use `linalg.solve(A, b)` where A is the coefficient matrix and b is the constant vector.

[1 Mark] How do you compute eigenvalues and eigenvectors?

Answer: Use `linalg.eig(A)` where A is a square matrix.

[1 Mark] Which module in SciPy deals with statistical functions?

Answer: `scipy.stats` provides functions for descriptive and inferential statistics.

[1 Mark] How can you calculate skewness and kurtosis of data?

Answer: Use `stats.skew(data)` and `stats.kurtosis(data)`.

[1 Mark] What does `stats.norm.pdf(x)` return?

Answer: It returns the probability density function (PDF) of a normal distribution at points x.

[1 Mark] What is the purpose of `scipy.signal`?

Answer: It provides functions for signal processing like filtering and spectral analysis.

[1 Mark] How do you apply a Butterworth filter using SciPy?

Answer: Use `butter()` to design the filter and `lfilter()` to apply it.

[1 Mark] What are special functions in SciPy?

Answer: Functions like gamma, Bessel, and elliptic integrals available via `scipy.special`.

[1 Mark] How can you compute the Gamma function in SciPy?

Answer: Use `special.gamma(x)`.

[1 Mark] What does `scipy.interpolate` provide?

Answer: It offers tools for interpolation of 1D and multi-dimensional data.

[1 Mark] How do you compute a Fast Fourier Transform in SciPy?

Answer: Use `fft()` from `scipy.fft` to compute the FFT of a signal.

[1 Mark] What is pandas in Python?

Answer: pandas is an open-source Python library used for data analysis and manipulation. It provides data structures like Series and DataFrame for handling structured data.

[1 Mark] What are the two core data structures in pandas?

Answer: The two core data structures are `Series` (1D labeled array) and `DataFrame` (2D labeled data table).

[1 Mark] How do you install pandas?

Answer: Use the command `pip install pandas` to install pandas.

[1 Mark] How do you import pandas in Python?

Answer: Use `import pandas as pd` to import pandas.

[1 Mark] How do you create a pandas Series?

Answer: Use `pd.Series(data, index)` where data is a list or array and index is a list of labels.

[1 Mark] What is a DataFrame in pandas?

Answer: A DataFrame is a two-dimensional, size-mutable, heterogeneous tabular data structure with labeled axes (rows and columns).

[1 Mark] How do you access a column in a DataFrame?

Answer: You can access a column using `df['ColumnName']`.

[1 Mark] What is the difference between `loc` and `iloc`?

Answer: `loc` accesses rows/columns by labels, whereas `iloc` accesses them by integer position.

[1 Mark] How do you perform boolean indexing in pandas?

Answer: You can filter data using conditions, e.g., `df[df['Age'] > 30]`.

[1 Mark] How do you add a new column to a DataFrame?

Answer: Assign a new column using `df['NewCol'] = values`.

[1 Mark] How do you modify a specific cell value in a DataFrame?

Answer: Use `df.at[row_index, column_label] = new_value`.

[1 Mark] How do you detect missing values in pandas?

Answer: Use `df.isnull()` to detect missing (NaN) values.

[1 Mark] How can missing values be handled in pandas?

Answer: Use `df.dropna()` to remove or `df.fillna(value)` to fill missing values.

[1 Mark] What is the purpose of `groupby` in pandas?

Answer: `groupby()` is used to group data by one or more columns and apply aggregation functions like `mean()`, `sum()`, etc.

[1 Mark] How do you concatenate DataFrames?

Answer: Use `pd.concat([df1, df2])` to concatenate along a particular axis.

[1 Mark] How do you merge two DataFrames in pandas?

Answer: Use `pd.merge(df1, df2, on='key')` to perform SQL-style joins.

[1 Mark] How do you read data from a CSV file using pandas?

Answer: Use `pd.read_csv('filename.csv')`.

[1 Mark] How do you write a DataFrame to an Excel file?

Answer: Use `df.to_excel('filename.xlsx', index=False)`.

[1 Mark] How does pandas support time series data?

Answer: pandas supports time series using `pd.date_range()` and time-based indexing on DataFrames.

[1 Mark] How can you visualize data using pandas?

Answer: You can use the `plot()` method in combination with matplotlib, e.g., `df['Salary'].plot(ki`

[1 Mark] What is Matplotlib?

Answer: Matplotlib is a Python 2D plotting library used to create static, interactive, and animated visualizations.

[1 Mark] How do you install Matplotlib?

Answer: You can install it using the command `pip install matplotlib`.

[1 Mark] Which module is commonly used for plotting in Matplotlib?

Answer: The `pyplot` module from `matplotlib` is most commonly used, typically imported as `import matplotlib.pyplot as plt`.

[1 Mark] How do you display a plot in Matplotlib?

Answer: Use `plt.show()` to display the plot window.

[1 Mark] How do you add a title and axis labels to a plot?

Answer: Use `plt.title()`, `plt.xlabel()`, and `plt.ylabel()`.

[1 Mark] How do you add a legend to a plot?

Answer: Use `plt.legend()` after labeling plot elements with the `label=` parameter.

[1 Mark] How can you plot multiple lines on the same figure?

Answer: Call `plt.plot()` multiple times before `plt.show()`.

[1 Mark] What are some common plot types supported by Matplotlib?

Answer: Line plots, bar charts, histograms, scatter plots, pie charts.

[1 Mark] How do you create a bar chart using Matplotlib?

Answer: Use `plt.bar(x, y)`.

[1 Mark] How do you create a histogram?

Answer: Use `plt.hist(data, bins)` to plot the frequency distribution of data.

[1 Mark] How do you create a scatter plot?

Answer: Use `plt.scatter(x, y)`.

[1 Mark] How do you create a pie chart?

Answer: Use `plt.pie()` and pass in values and optional labels.

[1 Mark] How can you customize a line's color, marker, and style?

Answer: Use parameters like `color=`, `marker=`, and `linestyle=` inside `plt.plot()`.

[1 Mark] How do you set limits on the x-axis and y-axis?

Answer: Use `plt.xlim()` and `plt.ylim()`.

[1 Mark] What is the use of `plt.grid(True)`?

Answer: It enables grid lines on the plot for better readability.

[1 Mark] What is the difference between `subplot()` and `subplots()`?

Answer: `subplot()` adds a subplot in a grid; `subplots()` creates a figure and axes

objects for multiple subplots.

[1 Mark] How do you save a plot to a file?

Answer: Use `plt.savefig("filename.png")` before calling `plt.show()`.

[1 Mark] What is the Object-Oriented interface in Matplotlib?

Answer: It's a more flexible plotting style where you create `Figure` and `Axes` objects using `fig, ax = plt.subplots()`.

[1 Mark] How does Matplotlib integrate with pandas?

Answer: pandas DataFrames support the `plot()` method which internally uses Matplotlib for visualization.

[1 Mark] Give an example use-case of Matplotlib in real-world data.

Answer: Plotting monthly sales figures to compare trends across years using line plots and bar charts.

[1 Mark] What is Seaborn?

Answer: Seaborn is a Python data visualization library based on Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics.

[1 Mark] How do you install Seaborn?

Answer: You can install it using the command `pip install seaborn`.

[1 Mark] Which module is typically imported to use Seaborn?

Answer: The Seaborn library is imported using `import seaborn as sns`.

[1 Mark] What types of data structures does Seaborn work best with?

Answer: Seaborn works best with pandas DataFrames.

[1 Mark] Name a built-in dataset available in Seaborn.

Answer: Examples include `"tips"`, `"titanic"`, and `"iris"`.

[1 Mark] What is the purpose of `sns.set_style()`?

Answer: It sets the visual theme of the plots. Options include `"white"`, `"dark"`, `"whitegrid"`, `"darkgrid"`, and `"ticks"`.

[1 Mark] How do you create a histogram with a density curve using Seaborn?

Answer: Use `sns.histplot(data=..., x=..., kde=True)`.

[1 Mark] What is a box plot used for in Seaborn?

Answer: A box plot shows the distribution, central value, and variability of a dataset.

[1 Mark] What is a violin plot and how is it different from a box plot?

Answer: A violin plot shows the full distribution of the data including density, unlike a box plot which shows summary statistics.

[1 Mark] What is the purpose of a strip plot?

Answer: It shows all individual data points along a categorical axis, useful for small datasets.

[1 Mark] What does `sns.barplot()` do?

Answer: It plots the mean (by default) of a numerical variable for each category of a categorical variable.

[1 Mark] What does `sns.countplot()` do?

Answer: It shows the counts of observations in each categorical bin using bars.

[1 Mark] How do you add color grouping to a scatter plot in Seaborn?

Answer: Use the `hue=` parameter in `sns.scatterplot()`.

[1 Mark] What is `sns.lmplot()` used for?

Answer: It plots a linear regression model fit along with a scatter plot.

[1 Mark] How do you create a heatmap of correlation values in Seaborn?

Answer: Use `sns.heatmap(corr, annot=True, cmap="coolwarm")` where `corr` is a correlation matrix.

[1 Mark] What does `sns.pairplot()` do?

Answer: It creates pairwise scatter plots for all numerical variables in the dataset.

[1 Mark] What is a FacetGrid in Seaborn?

Answer: A FacetGrid is used for drawing multiple plots based on row and column facets of a dataset.

[1 Mark] How can you customize Seaborn plots?

Answer: You can use `sns.set_palette()`, `plt.title()`, `plt.xlabel()`, and other Matplotlib functions.

[1 Mark] Can Seaborn be used with pandas groupby results?

Answer: Yes, you can aggregate with pandas and visualize with Seaborn, such as plotting group means.

[1 Mark] Give a real-world example of using Seaborn.

Answer: Visualizing survival rates by class and gender in the Titanic dataset using `sns.countplot()`.

[1 Mark] What is scikit-learn?

Answer: scikit-learn is a Python library for machine learning that provides tools for classification, regression, clustering, and more.

[1 Mark] How do you install scikit-learn?

Answer: You can install it using `pip install scikit-learn`.

[1 Mark] Name three types of tasks scikit-learn supports.

Answer: Classification, Regression, and Clustering.

[1 Mark] What is supervised learning?

Answer: Supervised learning involves learning from labeled data to make predictions or classifications.

[1 Mark] What is unsupervised learning?

Answer: Unsupervised learning involves learning patterns from data without labels.

[1 Mark] Name two supervised learning algorithms available in scikit-learn.

Answer: Logistic Regression and k-Nearest Neighbors (KNN).

[1 Mark] Which module provides built-in datasets like iris and digits?

Answer: `sklearn.datasets`.

[1 Mark] What is the use of `StandardScaler` in preprocessing?

Answer: It standardizes features by removing the mean and scaling to unit variance.

[1 Mark] How do you convert categorical labels to numerical form?

Answer: Using `LabelEncoder`.

[1 Mark] How do you split data into training and testing sets?

Answer: Using `train_test_split()` from `sklearn.model_selection`.

[1 Mark] Which metric is commonly used to evaluate classification performance?

Answer: Accuracy score.

[1 Mark] Which module provides the Logistic Regression model?

Answer: `sklearn.linear_model`.

[1 Mark] What is the purpose of `KNeighborsClassifier`?

Answer: It implements the k-Nearest Neighbors algorithm for classification.

[1 Mark] What does `mean_squared_error()` evaluate?

Answer: It measures the average squared difference between predicted and actual values in regression.

[1 Mark] Name one regression algorithm in scikit-learn.

Answer: Linear Regression.

[1 Mark] Which scikit-learn class is used for decision tree regression?

Answer: `DecisionTreeRegressor`.

[1 Mark] How do you perform clustering in scikit-learn?

Answer: Using models like `KMeans` from `sklearn.cluster`.

[1 Mark] What is cross-validation used for?

Answer: To evaluate model performance by splitting data into multiple folds and testing on different subsets.

[1 Mark] What is a pipeline in scikit-learn?

Answer: A pipeline chains multiple preprocessing steps and a model into one unit.

[1 Mark] How is PCA used in scikit-learn?

Answer: PCA is used for dimensionality reduction and visualization by projecting data onto principal components.

[1 Mark] What is Keras?

Answer: Keras is a high-level deep learning API written in Python that simplifies building and training neural networks.

[1 Mark] Which libraries can Keras run on top of?

Answer: TensorFlow, JAX, and PyTorch (via Keras Core).

[1 Mark] How do you install Keras?

Answer: Using `pip install keras` or `pip install tensorflow`.

[1 Mark] Name any two advantages of using Keras.

Answer: Easy-to-use API and support for multiple model architectures like CNNs and RNNs.

[1 Mark] What are the main steps in the typical Keras workflow?

Answer: Load data, define model, compile, train, evaluate, predict/save.

[1 Mark] Which built-in dataset is used in the example?

Answer: The MNIST dataset.

[1 Mark] What is the purpose of normalizing the dataset (e.g., dividing by 255.0)?

Answer: To scale the pixel values to the range [0, 1] for better model performance.

[1 Mark] How do you flatten a 2D image in Keras?

Answer: Using the `Flatten()` layer.

[1 Mark] Which activation function is commonly used in the output layer for classification?

Answer: `softmax`.

[1 Mark] How do you compile a Keras model?

Answer: Using `model.compile()` with optimizer, loss, and metrics.

[1 Mark] Which loss function is used for multi-class classification?

Answer: `sparse_categorical_crossentropy`.

[1 Mark] How do you train a Keras model?

Answer: Using `model.fit()` with training data and parameters like epochs and batch size.

[1 Mark] What method is used to evaluate a trained model?

Answer: `model.evaluate()`.

[1 Mark] How can you make predictions using a trained Keras model?

Answer: Using `model.predict()`.

[1 Mark] How do you save and load a Keras model?

Answer: `model.save()` to save, and `load_model()` to load.

[1 Mark] Name two layers commonly used in CNNs.

Answer: `Conv2D` and `MaxPooling2D`.

[1 Mark] What is the purpose of callbacks in Keras?

Answer: To monitor training, implement early stopping, logging, etc.

[1 Mark] What does the EarlyStopping callback do?

Answer: It stops training when the validation loss doesn't improve for a specified number of epochs.

[1 Mark] Which activation function is used for binary classification outputs?

Answer: `sigmoid`.

[1 Mark] Why is Keras suitable for beginners?

Answer: Its intuitive, high-level syntax makes it easy to build and train deep learning models quickly.

[1 Mark] What is TensorFlow?

Answer: TensorFlow is an open-source end-to-end machine learning platform developed by Google.

[1 Mark] Which high-level API does TensorFlow integrate tightly with?

Answer: `tf.keras`.

[1 Mark] Name two key features of TensorFlow.

Answer: Automatic differentiation and scalable training on CPU, GPU, and TPU.

[1 Mark] How do you install TensorFlow?

Answer: Using `pip install tensorflow`.

[1 Mark] What is a tensor in TensorFlow?

Answer: A tensor is a multi-dimensional array used as the basic unit of data in TensorFlow.

[1 Mark] How do you define a constant tensor?

Answer: Using `tf.constant()`.

[1 Mark] How do you define a variable tensor?

Answer: Using `tf.Variable()`.

[1 Mark] What is eager execution?

Answer: It allows operations to run immediately, returning concrete values without building graphs.

[1 Mark] Which module is used for automatic differentiation in TensorFlow?

Answer: `tf.GradientTape`.

[1 Mark] What does `tape.gradient()` return?

Answer: It returns the gradient of a function with respect to its input(s).

[1 Mark] How do you define a model using `tf.keras`?

Answer: By using the `Sequential()` API and adding layers.

[1 Mark] How is a model compiled in TensorFlow?

Answer: Using `model.compile()` with optimizer, loss, and metrics.

[1 Mark] What method is used to train a model?

Answer: `model.fit()`.

[1 Mark] How do you evaluate a trained model?

Answer: Using `model.evaluate()`.

[1 Mark] How can you save and load a model in TensorFlow?

Answer: Using `model.save()` and `tf.keras.models.load_model()`.

[1 Mark] What is a custom training loop?

Answer: It's a manual way to train a model using gradient tape and optimizer updates.

[1 Mark] How do you check if GPU is available in TensorFlow?

Answer: Using `tf.config.list_physical_devices('GPU')`.

[1 Mark] What is TensorBoard used for?

Answer: For visualizing training metrics like loss and accuracy.

[1 Mark] What dataset is used in the practical example?

Answer: The MNIST digit classification dataset.

[1 Mark] What is the use of `to_categorical()` in classification tasks?

Answer: It converts integer labels into one-hot encoded vectors.

[1 Mark] What is PyTorch?

Answer: PyTorch is an open-source deep learning framework developed by Facebook that supports tensor computation, dynamic graphs, and automatic differentiation.

[1 Mark] How do you install PyTorch?

Answer: Using `pip install torch torchvision`.

[1 Mark] What is a tensor in PyTorch?

Answer: A tensor is a multi-dimensional array similar to NumPy arrays but with GPU

support.

[1 Mark] How do you create a tensor filled with ones?

Answer: Use `torch.ones((rows, cols))`.

[1 Mark] How is matrix multiplication performed in PyTorch?

Answer: Using `torch.matmul(tensor1, tensor2)`.

[1 Mark] What is autograd in PyTorch?

Answer: Autograd is PyTorch's automatic differentiation engine used to compute gradients.

[1 Mark] How do you enable gradient computation on a tensor?

Answer: By setting `requires_grad=True` when defining the tensor.

[1 Mark] What module is used to define neural networks in PyTorch?

Answer: `torch.nn`.

[1 Mark] What is the role of `nn.Sequential`?

Answer: It defines a sequential container for stacking layers linearly.

[1 Mark] How is a loss function defined in PyTorch?

Answer: By creating an object like `nn.MSELoss()` or `nn.CrossEntropyLoss()`.

[1 Mark] How do you define an optimizer?

Answer: Use `torch.optim` classes like `SGD`, `Adam`, etc.

[1 Mark] What does `optimizer.zero_grad()` do?

Answer: It clears the gradients of all optimized tensors to prevent accumulation.

[1 Mark] How do you perform evaluation in PyTorch?

Answer: Use `model.eval()` and wrap code in `with torch.no_grad()`.

[1 Mark] How can you utilize GPU with PyTorch?

Answer: By moving tensors and models to the device using `.to(device)`.

[1 Mark] How do you save a PyTorch model?

Answer: Use `torch.save(model.state_dict(), 'filename.pth')`.

[1 Mark] How do you load a saved model in PyTorch?

Answer: First instantiate the model, then use `model.load_state_dict(torch.load('filename.pth'`
and call `model.eval()`.

[1 Mark] What is `DataLoader` used for?

Answer: It is used to load data in batches with optional shuffling and multiprocessing.

[1 Mark] What is the purpose of `transforms.ToTensor()`?

Answer: It converts PIL images or NumPy arrays to PyTorch tensors.

[1 Mark] Which dataset is used in the practical example?

Answer: The MNIST handwritten digit dataset.

[1 Mark] What is the output size of the final layer in the MNIST classifier example?

Answer: 10, corresponding to the 10 digit classes (0 to 9).

[1 Mark] What is OpenCV?

Answer: OpenCV (Open Source Computer Vision Library) is an open-source library for computer vision and machine learning tasks.

[1 Mark] How do you install OpenCV using pip?

Answer: By using `pip install opencv-python` or `pip install opencv-python-headless` for non-GUI environments.

[1 Mark] How do you read and display an image in OpenCV?

Answer: Use `cv2.imread()` to read and `cv2.imshow()` to display the image.

[1 Mark] How do you save an image using OpenCV?

Answer: Use `cv2.imwrite('filename.jpg', image)`.

[1 Mark] What does `img.shape` return?

Answer: It returns the height, width, and number of channels of the image.

[1 Mark] How do you resize an image in OpenCV?

Answer: Use `cv2.resize(image, (width, height))`.

[1 Mark] How can you crop an image?

Answer: By slicing the image array: `cropped = img[y1:y2, x1:x2]`.

[1 Mark] How do you draw a line in an image?

Answer: Use `cv2.line(img, pt1, pt2, color, thickness)`.

[1 Mark] Which function is used to write text on an image?

Answer: `cv2.putText()`.

[1 Mark] How do you convert an image to grayscale in OpenCV?

Answer: Use `cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`.

[1 Mark] What is the purpose of HSV color space in OpenCV?

Answer: HSV separates color information (hue) from intensity, useful for color detection and segmentation.

[1 Mark] How is thresholding done in OpenCV?

Answer: Using `cv2.threshold()` which converts grayscale images into binary format.

[1 Mark] What is Canny edge detection?

Answer: It detects edges in an image using gradients and thresholds with `cv2.Canny()`.

[1 Mark] What are common blurring techniques in OpenCV?

Answer: Gaussian blur and median blur, using `cv2.GaussianBlur()` and `cv2.medianBlur()`.

[1 Mark] What is the use of morphological operations like dilation and erosion?

Answer: Dilation expands white regions; erosion shrinks them, useful for removing noise or filling gaps.

[1 Mark] How do you find contours in an image?

Answer: Use `cv2.findContours()` on a binary image.

[1 Mark] What does `cv2.drawContours()` do?

Answer: It draws contours found by `cv2.findContours()` on the image.

[1 Mark] How do you capture video from a webcam in OpenCV?

Answer: Use `cv2.VideoCapture(0)` and read frames inside a loop.

[1 Mark] What is Haar cascade used for in OpenCV?

Answer: For object detection, such as detecting faces in real-time using pretrained classifiers.

[1 Mark] How do you detect and mark faces using OpenCV?

Answer: Use `face_cascade.detectMultiScale()` to detect and `cv2.rectangle()` to mark faces.