Python Toolkits

In Python, a **toolkit** generally refers to a collection of libraries and modules that provide the building blocks to solve specific programming problems or extend the language's capabilities into specialized fields. Examples: NumPy, Pandas, Matplotlib, Scikit-learn etc.

Introduction to NumPy

NumPy (short for **Numerical Python**) is a fundamental, open-source library for the Python programming language designed for scientific computing and numerical analysis. It provides a powerful N-dimensional array object, called **ndarray**, which is significantly faster and more memory-efficient than standard Python lists for handling large datasets.

Core Features

- **N-Dimensional Arrays (ndarray)**: The central data structure of NumPy, used to represent grids of values like vectors (1D), matrices (2D), or tensors (ND).
- **High Performance**: Unlike Python lists, NumPy arrays are stored in **contiguous memory** and are implemented largely in optimized **C code**, allowing for high-speed mathematical operations.
- **Vectorization**: Allows you to perform operations (like adding or multiplying) on entire arrays at once without writing explicit for loops, resulting in cleaner and faster code.
- **Broadcasting**: A mechanism that allows NumPy to perform arithmetic operations between arrays of different but compatible shapes.
- **Mathematical Library**: Includes a vast collection of functions for:
    - **Linear Algebra** (matrix multiplication, inversions).
    - **Statistics** (mean, median, standard deviation).
    - **Fourier Transforms** and **Random Number Generation**.

NumPy is considered the **backbone of the Python data science ecosystem**. Most major scientific libraries are built on top of it or interoperate with its array format, including:

Pandas: For data manipulation and analysis.

- SciPy: For advanced scientific and engineering algorithms.
- Matplotlib: For 2D/3D data visualization.

- Scikit-learn: For machine learning

NumPy Creating Arrays

NumPy is used to work with arrays. The array object in NumPy is called **ndarray**.

We can create a NumPy ndarray object by using the array() function.

Example

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

Output:

[1 2 3 4 5]

<class 'numpy.ndarray'>

To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:

Example

Use a tuple to create a NumPy array:

```
import numpy as np

arr = np.array((1, 2, 3, 4, 5))

print(arr)
```

NumPy Array Indexing

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

Example

Get the first element from the following array:

```
import numpy as np

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

print(arr[0])
```

Output: 1

NumPy Array Slicing

Slicing in python means taking elements from one given index to another given index.

We pass slice instead of index like this: [*start*:*end*].

We can also define the step, like this: [*start*:*end*:*step*].

If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1

Example

Slice elements from index 1 to index 5 from the following array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
```

Result: [2 3 4 5]

Note: The result *includes* the start index but *excludes* the end index.

NumPy Array Copy vs View

The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.

The copy *owns* the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy.

The view *does not own* the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

COPY:

Example :

Make a copy, change the original array, and display both arrays:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

Result:

[42 2 3 4 5]

[1 2 3 4 5]

The copy **should not** be affected by the changes made to the original array.

VIEW:

Example

Make a view, change the original array, and display both arrays:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
```

```
x = arr.view()
arr[0] = 42

print(arr)
print(x)
```

Result:

[42 2 3 4 5]

[42 2 3 4 5]

The view **should** be affected by the changes made to the original array.

NumPy Array Shape

The shape of an array is the number of elements in each dimension.

NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

Example

Print the shape of a 2-D array:

```
import numpy as np

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

print(arr.shape)
```

Result: (2, 4)

The example above returns (2, 4), which means that the array has 2 dimensions, where the first dimension has 2 elements and the second has 4.

NumPy Array Reshaping

Reshaping means changing the shape of an array.

The shape of an array is the number of elements in each dimension.

By reshaping we can add or remove dimensions or change number of elements in each dimension.

Reshape From 1-D to 2-D

Example

Convert the following 1-D array with 12 elements into a 2-D array.

The outermost dimension will have 4 arrays, each with 3 elements:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr)
```

Reshape From 1-D to 3-D

Example

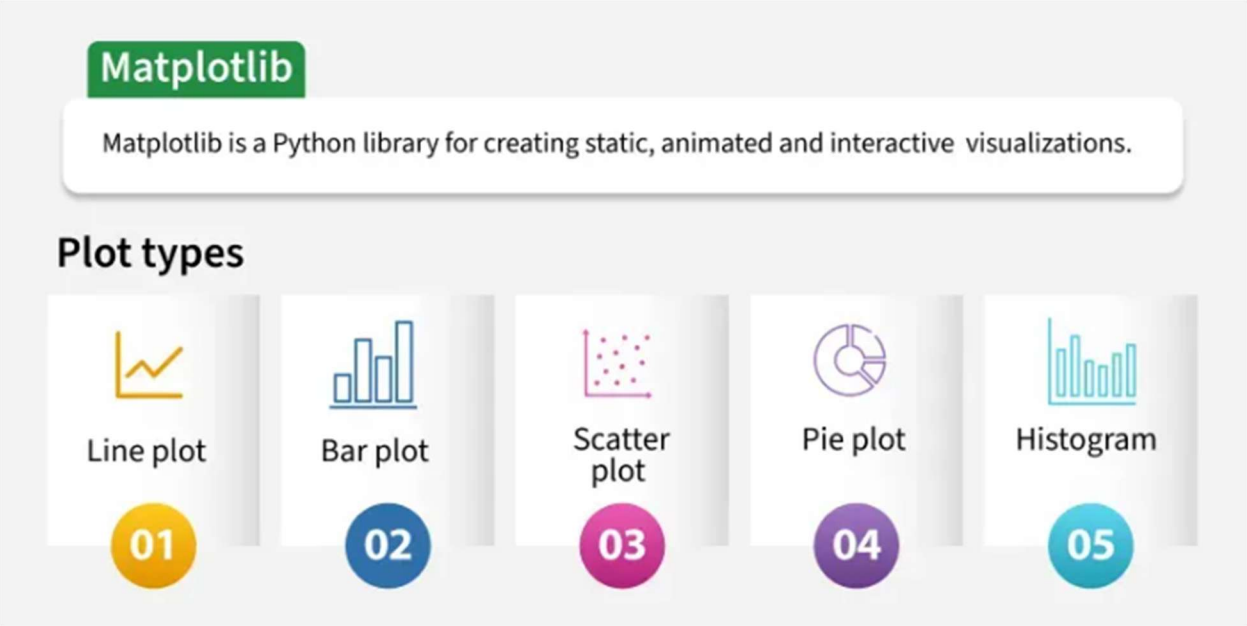
Convert the following 1-D array with 12 elements into a 3-D array.

The outermost dimension will have 2 arrays that contain 3 arrays, each with 2 elements:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(2, 3, 2)

print(newarr)
```

Matplotlib

Matplotlib is a widely used Python library for creating static, interactive and animated visualizations from data. It provides flexible and customizable plotting functions that help in understanding data patterns, trends and relationships effectively.



Different Types of Plots in Matplotlib

Matplotlib offers a wide range of plot types to suit various data visualization needs. Here are some of the most commonly used types of plots in Matplotlib:

- Line Graph
- Bar Chart
- Histogram
- Scatter Plot
- Pie Chart
- 3D Plot

Key Features of Matplotlib

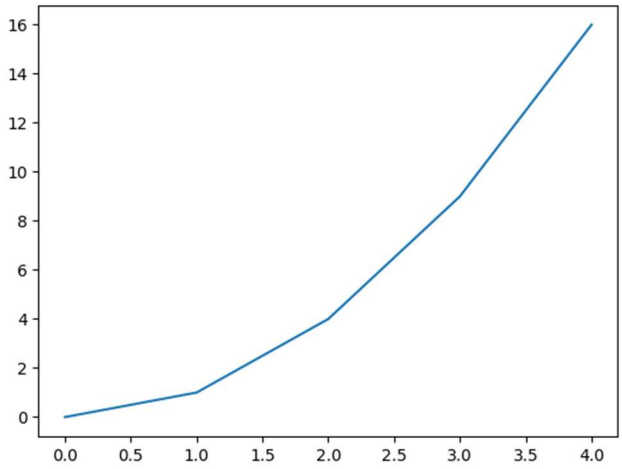
- **Versatile Plotting**: Create a wide variety of visualizations, including line plots, scatter plots, bar charts and histograms.
- **Extensive Customization**: Control every aspect of your plots, from colors and markers to labels and annotations.
- **Seamless Integration with NumPy**: Effortlessly plot data arrays directly, enhancing data manipulation capabilities.
- **High-Quality Graphics**: Generate publication-ready plots with precise control over aesthetics.
- **Cross-Platform Compatibility**: Use Matplotlib on Windows, macOS and Linux without issues.
- **Interactive Visualizations**: Engage with your data dynamically through interactive plotting features.

Matplotlib Pyplot

Pyplot is a module within Matplotlib that provides a MATLAB-like interface for making plots. It simplifies the process of adding plot elements such as lines, images and text to the axes of the current figure.

Example: Let's create a simple line plot using Matplotlib, showcasing the ease with which you can visualize data.

```
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4]

y = [0, 1, 4, 9, 16]

plt.plot(x, y)

plt.show()
```



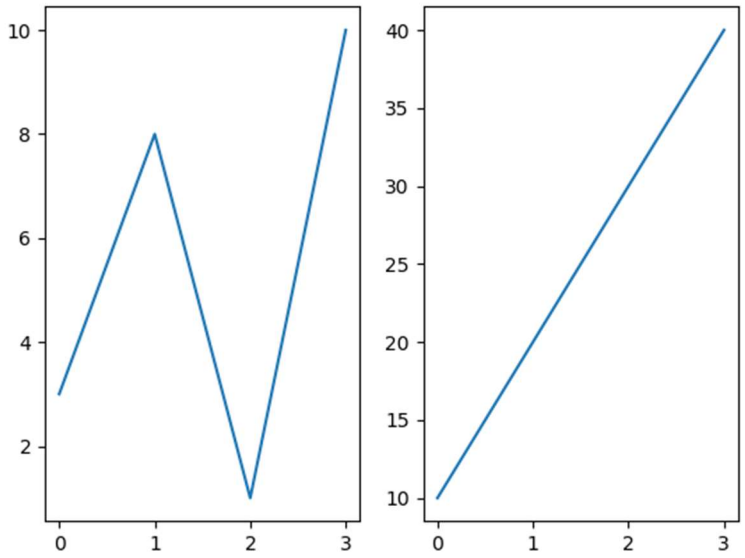
Matplotlib Subplot

With the **subplot()** function you can draw multiple plots in one figure:

Example

Draw 2 plots:

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

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)

plt.show()
```



Creating Scatter Plots

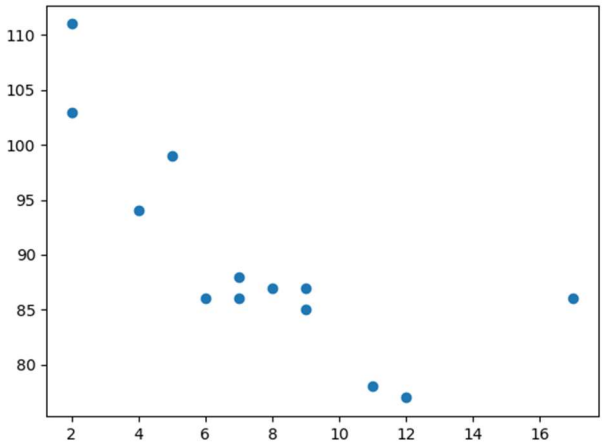
With Pyplot, you can use the scatter() function to draw a scatter plot.

The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

Example

A simple scatter plot:

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

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

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



The observation in the example above is the result of 13 cars passing by.

The X-axis shows how old the car is.

The Y-axis shows the speed of the car when it passes.

Are there any relationships between the observations?

It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

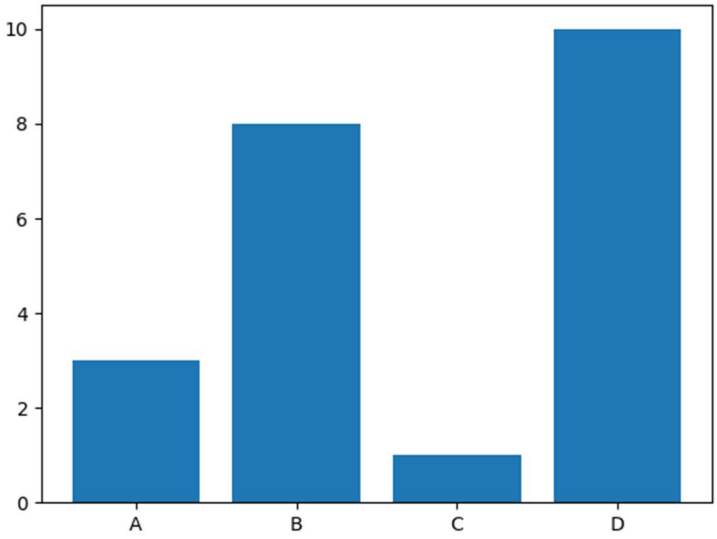
Creating Bar Chart

With Pyplot, you can use the bar() function to draw bar graphs:

Example

Draw 4 bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

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



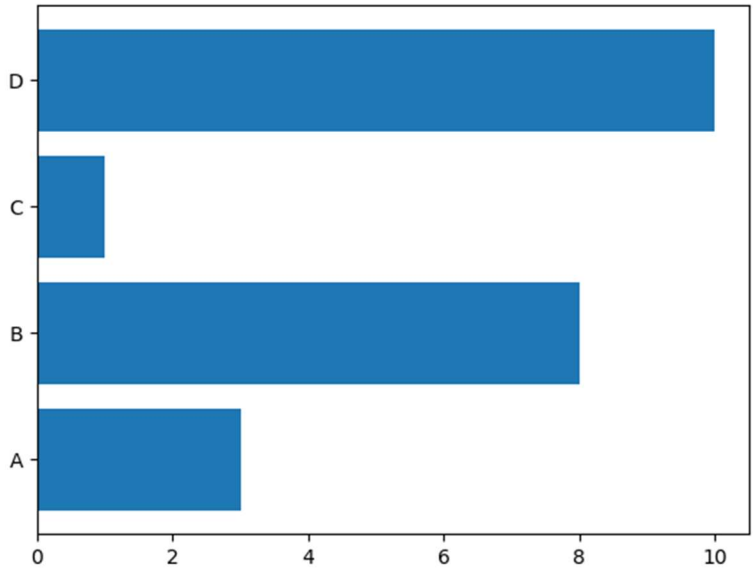
Horizontal Bars

If you want the bars to be displayed horizontally instead of vertically, use the barh() function:

Example

Draw 4 horizontal bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y)
plt.show()
```



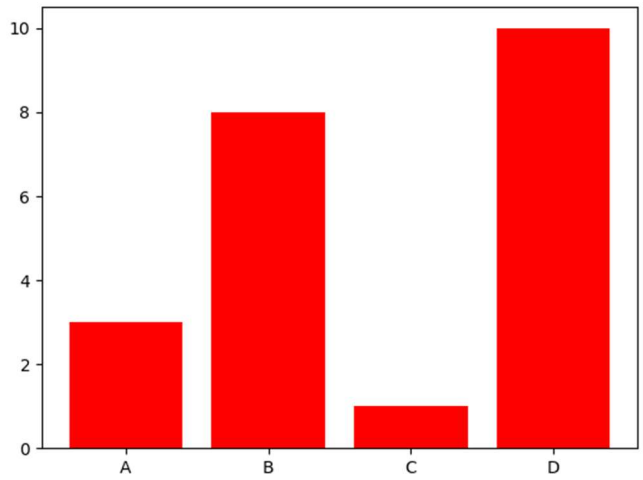
Bar Color

The bar() and barh() take the keyword argument color to set the color of the bars:

Example

Draw 4 red bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "red")
plt.show()
```

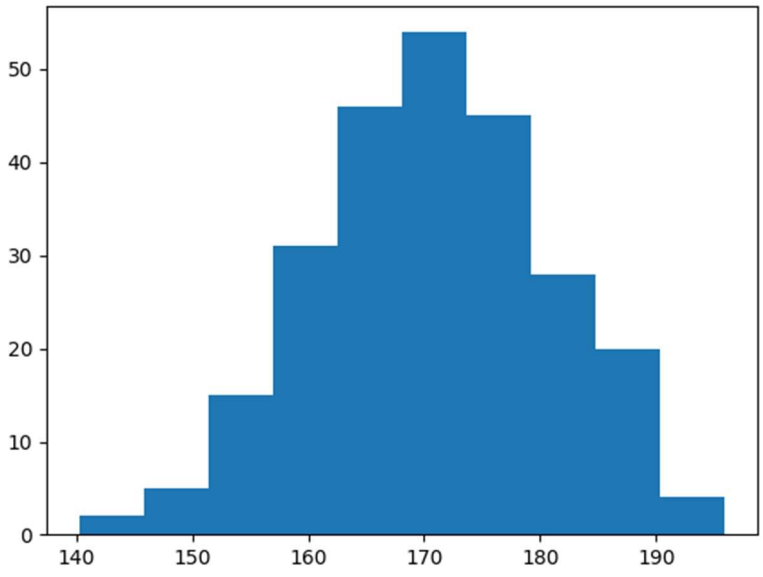


Histogram

A histogram is a graph showing *frequency* distributions.

It is a graph showing the number of observations within each given interval.

Example: Say you ask for the height of 250 people, you might end up with a histogram like this:



Create Histogram

In Matplotlib, we use the **hist()** function to create histograms.

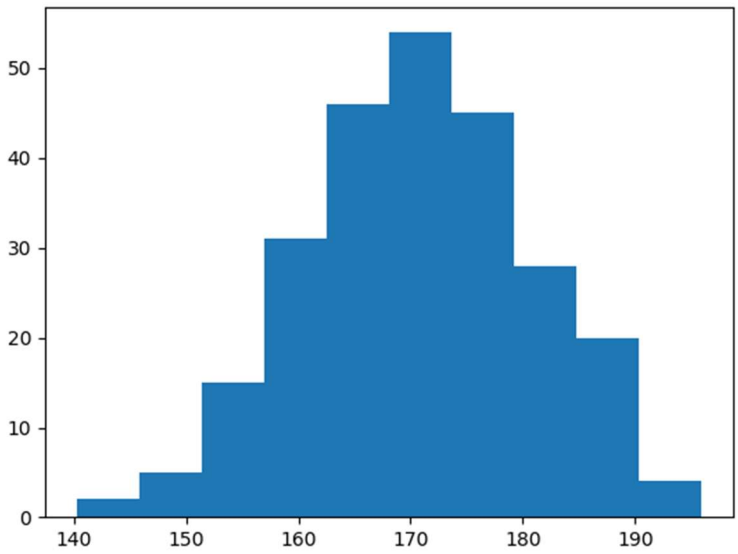
The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10.

Example

A simple histogram:

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

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```



Creating Pie Charts

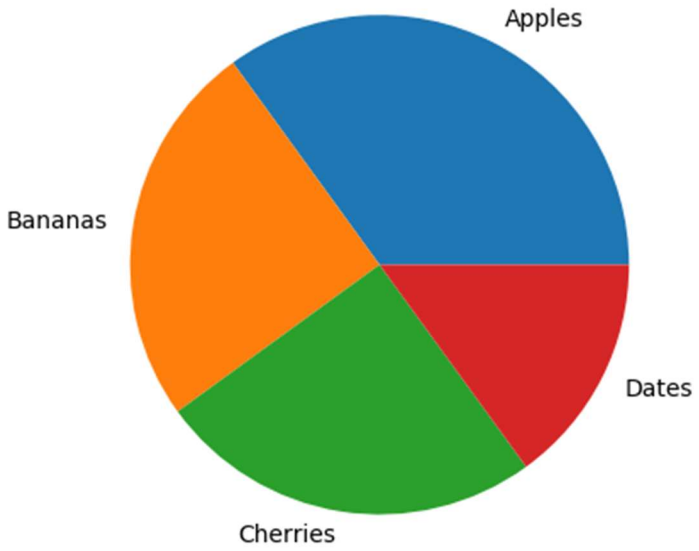
With Pyplot, you can use the pie() function to draw pie charts:

Example

A simple pie chart:

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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.show()
```

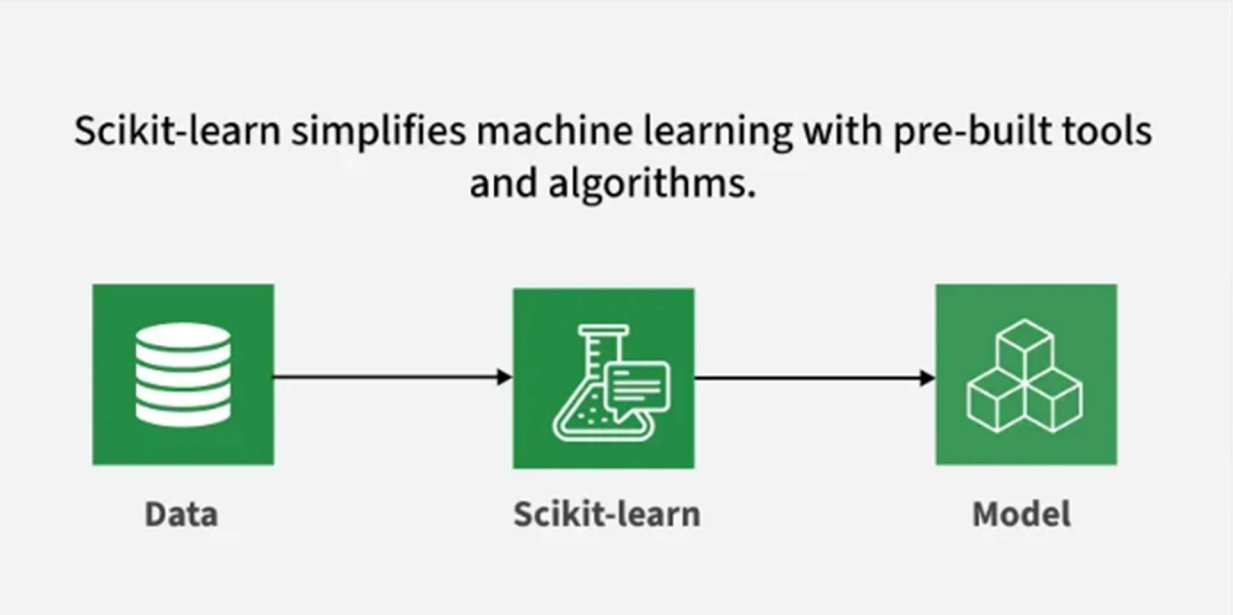
Result:

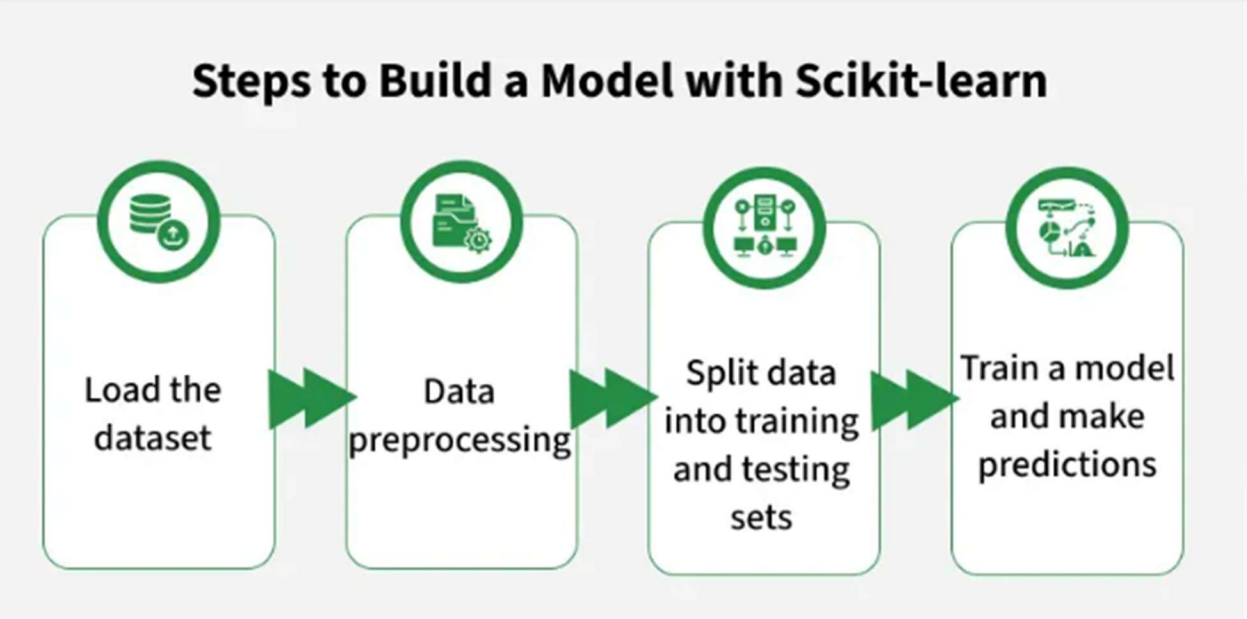


Scikit-learn

Scikit-learn is an open-source Python library that simplifies the process of building machine learning models. It offers a clean and consistent interface that helps both beginners and experienced users work efficiently.

- Supports tasks like classification, regression, clustering and preprocessing
- Makes model building fast and reliable
- Provides ready-to-use tools for training and evaluation
- Reduces complexity by avoiding manual implementation of algorithms





Features of Scikit-learn

Scikit-learn is used because it makes building machine learning models straightforward and efficient. Here are some important reasons:

1. **Ready-to-Use Tools**: It provides built-in functions for common tasks like data preprocessing, training models and making predictions. This saves time by avoiding the need to code algorithms from scratch.
2. **Easy Model Evaluation**: With tools like cross-validation and performance metrics it helps to measure how well our model works and identify areas for improvement.
3. **Wide Algorithm Support**: It offers many popular machine learning algorithms including classification, regression and clustering which gives us flexibility to choose the right model for our problem.
4. **Smooth Integration**: Built on top of important Python libraries like NumPy and SciPy so it fits into our existing data analysis workflow.
5. **Simple and Consistent Interface**: The same straightforward syntax works across different models helps in making it easier to learn and switch between algorithms.
6. **Model Tuning Made Easy**: Tools like grid search help us fine-tune our model's settings to improve accuracy without extra hassle.

Benefits of using Scikit-learn

- **User-Friendly**: Scikit-learn's consistent and simple interface makes it accessible for beginners and best for experts.
- **Time-Saving:** Pre-built tools and algorithms reduce development time which allows us to focus more on solving problems than coding details.
- **Better Model Performance**: Easy-to-use tuning and evaluation tools helps in improving model accuracy and reliability.
- **Flexible and Scalable**: Supports a wide range of algorithms and integrates smoothly with other Python libraries helps in making it suitable for projects of any size.
- **Strong Community Support**: A large, active community ensures regular updates, extensive documentation and plenty of resources to help when we get stuck.

NLTK

Natural Language Processing (NLP) plays an important role in enabling machines to understand and generate human language. Natural Language Toolkit (NLTK) stands out as one of the most widely used libraries. It provides a combination linguistic resources, including text processing libraries and pre-trained models, which makes it ideal for both academic research and practical applications.

NLTK is a Python's API library and it can perform a variety of operations on textual data such as classification, tokenization, stemming, tagging, semantic reasoning, etc.

NLTK

NLTK is a versatile library designed to simplify natural language proces-sing with functionalities including text tokenization, tag, and word frequency counting.

01 Tokenize Text into Words

02 Segment Words into Syllabus

03 Determine Parts of Speech

04 Calculate Word Frequencies

Applications of NLTK

01 Sentiment analysis in social media

02 Information extraction

03 Language learning and teaching

04 Machine translation

05 Chatbots and virtual assistants

06 Text summarization

Some basic operations on text data using NLTK :

1. Tokenization

Tokenization refers to break down the text into smaller units. It splits paragraphs into sentences and sentences into words. It is one of the initial steps of any NLP pipeline. Let us have a look at the two major kinds of tokenization that NLTK provides:

1.1 Word Tokenization

It involves breaking down the text into words.

"I study Machine Learning on googlecolab ."
will be word-tokenized as:

['I', 'study', 'Machine', 'Learning', 'on', 'googlecolab', '.']

1.2 Sentence Tokenization

It involves breaking down the text into individual sentences.

"I study Machine Learning on googlecolab. Currently, I'm studying NLP"
will be sentence-tokenized as :

['I study Machine Learning on googlecolab.', 'Currently, I'm studying NLP.']

.

2. Stemming and Lemmatization

When working with natural language, our focus is on understanding the intended meaning behind words. To achieve this, it is essential to reduce words to their root or base form. This process is known as **canonicalization**.

There are two commonly used techniques for canonicalization: **stemming** and **lemmatization**.

2.1 Stemming

Stemming generates the base word from the given word by removing the affixes of the word. It has a set of pre-defined rules that guide the dropping of these affixes. It must be noted that stemmers might not always result in semantically meaningful base words. Stemmers are faster and computationally less expensive than lemmatizers.

2.2 Lemmatization

Lemmatization involves grouping together the inflected forms of the same word. This way, we can reach out to the base form of any word which will be meaningful in nature. The base from here is called the Lemma.

Lemmatizers are slower and computationally more expensive than stemmers.

Working with data: Reading Files

In data science, the primary method for reading files is using the pandas library in Python, which efficiently handles various data formats and structures.

Using Pandas

Pandas is the recommended library for handling structured data as it loads files into a DataFrame, an easy-to-use data structure.

- **CSV (Comma-Separated Values):** The most common format.
  The read_csv() function is used, and it is highly flexible, supporting different delimiters with the sep argument (e.g., a tab character '\t' for TSV files).

```
python

import pandas as pd

df = pd.read_csv("filename.csv")
```

- **Excel (XLSX, XLS, ODS):** Pandas can read Excel files. For .xlsx files, you need optional dependencies like openpyxl.

python

import pandas as pd

 # To read a specific sheet:

df = pd.read_excel("filename.xlsx", sheet_name="Sheet1")

- **JSON (JavaScript Object Notation):** A standard format for structured data exchange over the web.

python

import pandas as pd

df = pd.read_json("filename.json")

- **Other formats:** Pandas also support reading from databases, HTML tables, HDF5, Parquet, and more.

Key Steps for Reading Files

- Regardless of the method, the process generally follows these steps:
- **Specify the path** to the data (either an absolute path, relative path, or URL).
- **Open a connection** or use a library function to access the file.
- **Read the data**, potentially in chunks for very large files to manage memory usage.
- **Process and store** the data in an appropriate structure (like a pandas DataFrame or a list).
- **Close the connection** to the file (handled automatically by the with statement).

Data Munging

Data munging, sometimes called data wrangling or data cleaning, is converting and mapping unprocessed data into a different format to improve its suitability and value for various downstream uses, including analytics. This procedure entails preparing raw data for analysis by cleaning, organizing, and enriching it in a readable format.

Why is Data Munging important?

Data munging holds immense significance in the field of data analysis, playing a crucial role in ensuring the quality and reliability of the data used for making informed decisions. Several key aspects highlight the significance of data munging in the data analysis process:

- **Accuracy and Precision:** Data munging addresses discrepancies and errors in raw data, leading to more accurate and precise analyses. Cleaning and organizing data ensure that the insights derived are trustworthy and dependable.

- **Quality Improvement**: By cleaning and preprocessing data, errors and inconsistencies are reduced, improving overall data quality.
- **Compatibility**: Data munging ensures that data from different sources can be integrated and used together effectively.
- **Analysis Readiness**: Properly formatted data is essential for accurate analysis and modeling, enabling organizations to make data-driven decisions.
- **Facilitation of Decision-Making**: Clean and well-structured data, resulting from effective data munging, facilitates the decision-making process. Decision-makers can rely on accurate insights derived from trustworthy data.

Essential Steps in Data Munging

here are the different stages of data munging:

1. **Data Discovery**:
    - Defining the purpose and goals of data analysis.
    - Identifying potential uses and requirements of data.
    - Focusing on business requirements rather than technical specifications.
2. **Data Structuring**:
    - Structuring raw data to make it machine-readable.
    - Organizing data into a well-defined schema with consistent layout (rows and columns).
    - Extracting data from various sources and organizing it into a formatted repository.
3. **Data Cleansing**:
    - Addressing data quality issues such as missing values and duplicate datasets.
    - Detecting and correcting erroneous data to avoid information gaps.
    - Applying transformations (e.g., removing, replacing, finding and replacing) to eliminate redundant text and null values, and identify missing fields and typing errors.
    -

4. **Data Enrichment**:
    - Appending one or multiple datasets from different sources to generate a holistic view of information.
    - Aggregating multiple data sources to make data more useful for reporting and analytics.
    - Example: Matching an order ID against a different database to obtain further details like account name, account balance, buying history, etc.
5. **Data Validation**:
    - Validating the accuracy, completeness, and reliability of data.
    - Final check to ensure output information is accurate and reliable.
    - Rejecting data that don't comply with pre-defined rules or constraints.
    - Types of validation checks include consistency check, data-type validation, range and constraint validation.

These stages represent a systematic approach to preparing data for analysis, ensuring that the data is well-structured, clean, enriched, and validated before further analysis or processing. Each stage plays a crucial role in the overall data munging process, ultimately leading to more accurate and reliable insights from data analysis.

Data Rescaling

Data rescaling is a fundamental data preprocessing step in machine learning and statistics that involves transforming numeric input variables to a specific, common range or distribution. It ensures that features with large magnitudes do not disproportionately dominate the model's learning process, which is particularly important for distance-based algorithms like k-NN, SVM, or those using gradient descent.

Why Rescale Data?

- **Improved Model Performance:** Prevents features with large ranges (e.g., income: $0–$100,000) from dominating features with small ranges (e.g., age: 0–100), ensuring equal importance.
- **Faster Convergence:** Helps gradient-based optimization methods (e.g., neural networks) reach a solution faster and more stably.

- **Avoids Numerical Instability:** Prevents algorithms from failing due to extremely large values.
- **Algorithm Requirements:** Some algorithms, such as PCA or KNN, assume data is centered around zero or scaled similarly

Common Data Rescaling Techniques

1. **Min-Max Scaling (Normalization)**

   Description: Rescales the data to a fixed range, usually 0 to 1.

   $$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}.$$

   Best Use: When you know the minimum and maximum boundaries of your data and the data does not follow a Gaussian distribution.

   Drawback: Sensitive to outliers.

2. **Standardization (Z-score Normalization)**

   Description: Centers the data around a mean of 0 with a standard deviation of 1.

   Formula: $X_{new} = \frac{X - \mu}{\sigma}$ (where μ is mean and σ is standard deviation).

   Best Use: When data follows a Gaussian distribution or when using algorithms like Linear Regression or Logistic Regression.

   Benefit: More robust to outliers than Min-Max scaling.

3. **Log Transformation**

Description: Log scaling is a normalization technique that is useful when the data has a skewed distribution. This technique involves taking the logarithm of the data to reduce the effect of extreme values.

Example: Suppose we have a dataset with a feature "Income" that has a skewed distribution. To normalize this feature using log scaling, we would take the logarithm of each income value. This would result in a normalized feature with a more even distribution.

Dimensionality Reduction

When working with machine learning models, datasets with too many features can cause issues like slow computation and overfitting. Dimensionality reduction helps to reduce the number of features while retaining key information. It converts high-dimensional data into a lower-dimensional space while preserving important details.

Dimensionality Reduction Techniques

Dimensionality reduction techniques can be broadly divided into two categories:

1. Feature Selection

Feature selection chooses the most relevant features from the dataset without altering them. It helps remove redundant or irrelevant features, improving model efficiency. Some common methods are:

- Filter methods rank the features based on their relevance to the target variable.
- Wrapper methods use the model performance as the criteria for selecting features.
- Embedded methods combine feature selection with the model training process.

2. Feature Extraction

Feature extraction involves creating new features by combining or transforming the original features. These new features retain most of the dataset's important information in fewer dimensions. Common features extraction methods are:

1. **Principal Component Analysis (PCA):** Converts correlated variables into uncorrelated principal components hence reducing dimensionality while maintaining as much variance as possible enabling more efficient analysis.
2. **Missing Value Ratio:** Variables with missing data beyond a set threshold are removed, improving dataset reliability.
3. **Backward Feature Elimination**: Starts with all features and removes the least significant ones in each iteration. The process continues until only the most impactful features remain, optimizing model performance.
4. **Forward Feature Selection:** It begins with one feature, adds others incrementally and keeps those improving model performance.

5. **Random Forest**: Random forest uses decision trees to evaluate feature importance, automatically selecting the most relevant features without the need for manual coding, enhancing model accuracy.
6. **Factor Analysis**: Groups variables by correlation and keeps the most relevant ones for further analysis.
7. **Independent Component Analysis (ICA)**: Identifies statistically independent components, ideal for applications like 'blind source separation' where traditional correlation-based methods fall short.