How to... Python Packages

Introduction

Python inherently offers useful functionalities such as `print()`, `enumerate()`, or `list.append()`. However, our projects often require more specific tools. This is where packages (or libraries) come in. Packages are extensions for Python that provide specialized functions and tools to efficiently solve a variety of problems.

The most frequently used packages include:

- `numpy`: For fast numerical and mathematical computations.
- `pandas`: For the efficient processing of large amounts of structured data (tables).
- `matplotlib`: For creating data visualizations and diagrams.

Instead of treating every function of these packages in detail, which would be too extensive, this summary focuses on practical application examples for each presented package.

NumPy

Transposing a matrix means swapping rows and columns. If A is a matrix, then its transposed matrix A^T is defined by:

$$(A^T)_{ij} = A_{ji}$$

If one were to perform a matrix transposition with pure Python, one would have to manually iterate over rows and columns, leading to comparatively inefficient code.

```python
matrix = [[1, 2], [3, 4]]
rows = len(matrix)
cols = len(matrix[0])
transposed = []

for i in range(cols):
    new_row = []
    for j in range(rows):
        new_row.append(matrix[j][i])
    transposed.append(new_row)

print(transposed)
```

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

Depending on the size of the matrix, this process can be very time-consuming. With NumPy, we can use the optimized transposition function to solve this problem quickly and elegantly. NumPy operations are vectorized and run with significantly better performance.

```python
import numpy as np

matrix = np.array([[1, 2], [3, 4]])
transposed = matrix.T

print(transposed)
```

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

> Tip
>
> Check out the Numpy Cheat Sheet.

Pandas

Working with structured data in native Python is possible, but quickly becomes complicated and error-prone with larger data sets. Even simply summarizing a table already requires a considerable amount of code and manual data processing.

```
sum_col_a = 0.0
sum_col_b = 0.0
count = 0

with open("data/pandasTable.csv", "r") as f:
    header = f.readline()

    for line in f:
        values = line.strip().split(',')
        sum_col_a += float(values[1])
        sum_col_b += float(values[2])
        count += 1

avg_col_a = 0.0
avg_col_b = 0.0

if count > 0:
    avg_col_a = sum_col_a / count
    avg_col_b = sum_col_b / count

print("Number of Records:", count)
print("Column ColA:")
print("  Sum:", sum_col_a)
print("  Average:", avg_col_a)
print("Column ColB:")
print("  Sum:", sum_col_b)
print("  Average:", avg_col_b)
```

```
Number of Records: 5
Column ColA:
  Sum: 82.2
  Average: 16.44
Column ColB:
  Sum: 805.0
  Average: 161.0
```

With Pandas, we can easily read in structured data and retrieve summary statistics with a single function call, which simplifies data analysis considerably.

```
import pandas as pd
```

```
df = pd.read_csv("data/pandasTable.csv")
df.describe()
```

	ID	hight	width	Group
count	5.000000	5.000000	5.000000	5.00000
mean	3.000000	16.440000	161.000000	1.80000
std	1.581139	10.582911	108.880669	0.83666
min	1.000000	5.900000	50.000000	1.00000
25%	2.000000	10.500000	100.000000	1.00000
50%	3.000000	12.700000	125.000000	2.00000
75%	4.000000	20.100000	200.000000	2.00000
max	5.000000	33.000000	330.000000	3.00000

> **Tip**
>
> Check out the Pandas Cheat Sheet.

Matplotlib

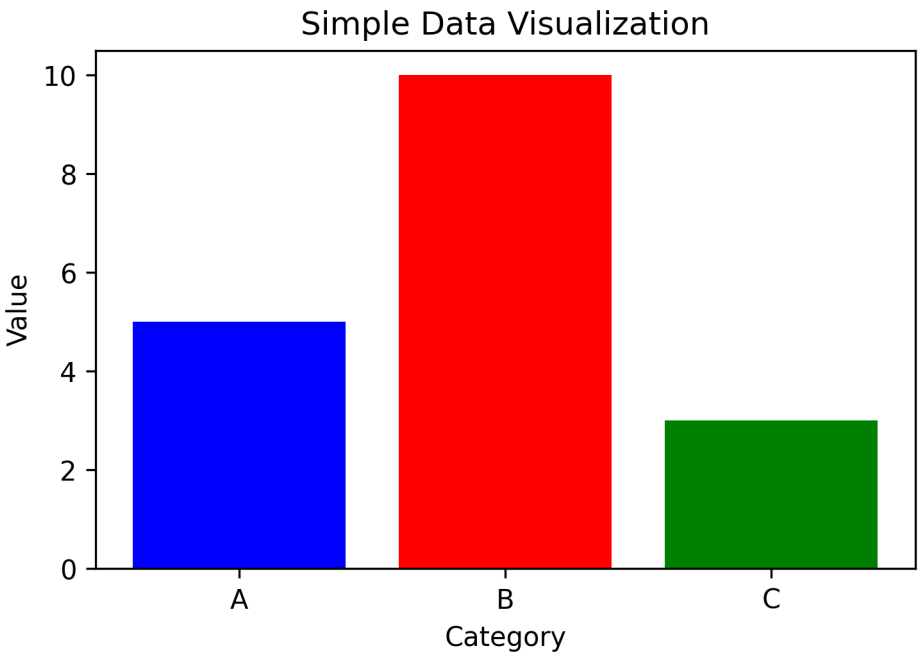
Data visualization in pure Python is a challenge, as the language was not designed for drawing diagrams without specialized packages.

```
data = {"A": 5, "B": 10, "C": 3}
max_val = max(data.values())
scale = 50 / max_val  # Skalierung auf max. 50 Zeichen Breite

for key, value in data.items():
    bar_length = int(value * scale)
    print(f"{key}: {'#' * bar_length} {value}")
```

```
A: ########################## 5
B: ################################################## 10
C: ################ 3
```

Fortunately, Matplotlib provides us with a versatile library with which we can create various types of diagrams.

```python
import matplotlib.pyplot as plt

data = {"A": 5, "B": 10, "C": 3}

keys = list(data.keys())
values = list(data.values())

plt.figure()
plt.bar(keys, values, color=['blue', 'red', 'green'])

plt.title('Simple Data Visualization')
plt.xlabel('Category')
plt.ylabel('Value')

plt.show()
```



> **Tip**
>
> Check out the Matplotlib Cheat Sheet.

More to explore

Thousands of packages are available for Python. However, it is advisable to use well-maintained and established standard packages such as `numpy`, `pandas`, and `matplotlib`, as these are regularly updated and have large communities.

If you are looking for a specific tool, PyPI (Python Package Index) is the central hub: PIPY