Lecture 06 - Libraries

Overview

`Python libraries` are **collections of modules and functions** that provide additional functionality, allowing to perform complex tasks with minimal code.

In this notebook covers:

- Why Python libraries are important
- How to install and import libraries
- Key Python libraries used in banking and finance
    - `pandas` for data manipulation
    - `numpy` for numerical computations
    - `matplotlib` and `seaborn` for data visualization
    - `datetime` for date and time manipulation

1. Why Use Python Libraries?

Python libraries allow to:

- Reuse code that has been tested and optimized by others
- Perform complex tasks with just a few lines of code
- Simplify workflow by providing tools tailored to specific tasks

Example: Using a Library vs. Writing Code from Scratch

In [ ]:

```
# Without using a library
# Calculate the square root of a number (writing the logic from scratch
def sqrt(number):
    return number ** 0.5

print("Square root of 16 (without library):", sqrt(16))
```

In [ ]:

```
# Using a library (numpy)
import numpy as np
print("Square root of 16 (using numpy):", np.sqrt(16))
```

2. Installing and Importing Libraries

To use a library, it needs to be installed (if it's not already installed) and be imported it into the current Python environment.

2.1 Installing Libraries

- Using `pip`, the Python package installer. For example:

```
pip install pandas numpy matplotlib seaborn
```

- Using the anaconda suite to install manually.

2.2 Importing Libraries

Once installed, libraries can be imported into the environment using `import (...as)`:

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

3. Key Python Libraries for Data Science in Finance

3.1 `numpy` for Numerical Computations

`numpy` is the foundational package for numerical computing in Python. It is particularly useful for handling large arrays and matrices of numeric data.

```
In [ ]: import numpy as np

        # Example: Calculating compound interest using numpy
        principal = 1000  # Initial amount
        rate = 0.05  # Interest rate
        time = 10  # Number of years

        # Calculate the future value
        future_value = principal * np.power(1 + rate, time)
        print("Future Value after 10 years:", future_value)
```

3.2 `pandas` for Data Manipulation

`pandas` is a powerful library for data manipulation and analysis. It works with structured data, such as CSV files, databases, and Excel spreadsheets.

In [ ]:

```python
import pandas as pd

# Example: Reading a CSV file containing transaction data
transactions = pd.DataFrame({
    'Date': ['2023-08-01', '2023-08-02', '2023-08-03'],
    'Description': ['Deposit', 'Withdrawal', 'Deposit'],
    'Amount': [1000, -200, 500]
})

# Display the first few rows of the DataFrame
print(transactions.head())
print ("\n")
# Calculating the total balance after transactions
transactions['Balance'] = transactions['Amount'].cumsum()
print(transactions)
```

3.3 `matplotlib` and `seaborn` for Data Visualization

`matplotlib` and `seaborn` are powerful libraries for creating visualizations. `matplotlib` provides a flexible foundation, while `seaborn` simplifies the creation of more complex plots.

In [ ]:

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

# Example: Visualizing the account balances over time
sns.lineplot(x='Date', y='Balance', data=transactions)
plt.title("Account Balance Over Time")
plt.xlabel("Date")
plt.ylabel("Balance")
plt.xticks(rotation=45)
plt.show()
```

In [ ]:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Sample Data: Tips Dataset (comes built-in with seaborn)
data = sns.load_dataset("tips")

# Create a rich visualization using Seaborn's relplot
sns.set_theme(style="whitegrid")  # Set theme for aesthetics
plot = sns.relplot(
    data=data,
    x="total_bill",
    y="tip",
    hue="day",
    style="time",
    size="size",
    palette="viridis",
    kind="scatter",
    aspect=1.5,
    height=6
)

# Add labels and a title
plot.set_axis_labels("Total Bill ($)", "Tip ($)")
plot.fig.suptitle("Tips vs. Total Bill by Day and Time", fontsize=16, w

# Show the plot
plt.show()
```

In [ ]:

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

# Load the built-in 'iris' dataset from seaborn
iris = sns.load_dataset("iris")

# Create a pairplot with rich formatting
sns.set(style="ticks", palette="pastel")
# pairplot() visualizes pairwise relationships in a dataset, making it
plot = sns.pairplot(
    iris,
    hue="species",  # Color by species type
    kind="reg",     # Add regression lines to show trends
    diag_kind="kde",  # Use KDE plots on the diagonal for a smoother di
    markers=["o", "s", "D"],  # Use different markers for each species
    height=2.5
)

# Add a main title and adjust spacing
plot.fig.suptitle("Pairplot of Iris Dataset with Regression Lines", for
plot.fig.subplots_adjust(top=0.95)

# Display the plot
plt.show()
```

3.4 `datetime` for Date and Time Manipulation

The `datetime` module allows you to work with dates and times.

In [ ]:

```
from datetime import datetime

# Example: Calculating the number of days between two dates
date_format = "%Y-%m-%d"
start_date = datetime.strptime("2023-08-01", date_format)
end_date = datetime.strptime("2023-08-10", date_format)

days_between = (end_date - start_date).days
print(f"Days between {start_date.date()} and {end_date.date()}: {days_b
```