Data Wrangling1

Overview

Data wrangling is the process of transforming raw data into a format that is suitable for analysis. It involves cleaning, structuring, and enriching data to facilitate data analysis and visualization. This module covers key topics, tools, and techniques essential for effective data wrangling.

---

1. Data Sources

Understanding the different types of data sources is crucial for effective data wrangling. Common sources include:

- **Text Files**: Files like CSV, TSV, and JSON are human-readable and store data in structured or semi-structured formats. They are easy to manipulate and widely used for data exchange.
- **Binary Files**: Formats like Excel, Parquet, and HDF5 store data in compact binary form, making them efficient for large datasets.
- **Web**: Data can be extracted from web pages through scraping or accessed via web APIs that provide structured data.
- **Databases**: SQL databases (e.g., MySQL, PostgreSQL) are relational and use structured queries. NoSQL databases (e.g., MongoDB) are more flexible, handling unstructured or semi-structured data.

---

2. Data Loading, Storage, and File Formats

Reading and Writing Data in Text Format

Data in text formats like CSV or JSON can be easily imported and exported using Python libraries.

- **CSV (Comma-Separated Values)**: Stores tabular data as plain text, with columns separated by commas.
- **JSON (JavaScript Object Notation)**: Represents structured data as key-value pairs, suitable for hierarchical data.

Example: Reading CSV Files

```
import pandas as pd

# Reading a CSV file
data = pd.read_csv('data.csv')
print(data.head())

# Writing to a CSV file
data.to_csv('output.csv', index=False)
```

Example: Reading JSON Files

```
import json

# Reading JSON
data = json.load(open('data.json'))
print(data)

# Writing JSON
with open('output.json', 'w') as f:
    json.dump(data, f)
```

---

3. Web Scraping

Web scraping involves extracting data from websites. It is useful for collecting data that is not readily available through APIs.

- **HTML Parsing**: Extracts specific elements from web pages using libraries like `BeautifulSoup`.
- **HTTP Requests**: Fetches web pages using the `requests` library.

Example: Scraping a Web Page

```
import requests
from bs4 import BeautifulSoup

# Fetching the web page
url = "https://example.com"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

# Extracting data
data = soup.find_all('h2')
for item in data:
    print(item.text)
```

---

4. Binary Data Formats

Binary formats are compact and efficient for storing large datasets.

- **Excel Files**: Widely used for spreadsheets, can handle complex formatting and formulas.
- **Parquet**: Optimized for fast read/write operations and suitable for big data frameworks.

Example: Working with Excel Files

```
import pandas as pd

# Reading Excel files
data = pd.read_excel('data.xlsx')
print(data.head())

# Writing to Excel
data.to_excel('output.xlsx', index=False)
```

Example: Working with Parquet Files

```
# Parquet
import pandas as pd

data = pd.read_parquet('data.parquet')
data.to_parquet('output.parquet')
```

---

5. Interacting with Web APIs

APIs (Application Programming Interfaces) allow structured access to external data sources over the web.

- **REST APIs**: Use HTTP methods (GET, POST, PUT, DELETE) to interact with resources.
- **Authentication**: Many APIs require API keys or OAuth tokens for secure access.

Example: Using an API

```
import requests

# API call
url = "https://api.example.com/data"
response = requests.get(url)
data = response.json()
print(data)
```

---

6. Interacting with Databases

SQL Databases

SQL databases store structured data in tables and use SQL (Structured Query Language) for queries.

- **Connection**: Requires database credentials (host, username, password).
- **Query Execution**: Retrieve or manipulate data using SQL commands.

Example: Connecting to MySQL

```
import mysql.connector

# Connecting to the database
conn = mysql.connector.connect(
    host='localhost',
    user='username',
    password='password',
    database='database_name'
)

# Query execution
cursor = conn.cursor()
cursor.execute("SELECT * FROM table_name")
for row in cursor.fetchall():
    print(row)

conn.close()
```

---

7. Data Wrangling Techniques

Hierarchical Indexing

Hierarchical indexing allows multi-level row or column indexing for better data organization.

Example:

```
import pandas as pd

# MultiIndex DataFrame
data = pd.DataFrame({
```

```
    'City': ['NY', 'NY', 'LA', 'LA'],
    'Year': [2020, 2021, 2020, 2021],
    'Value': [100, 150, 200, 250]
}).set_index(['City', 'Year'])

print(data)
```

Combining and Merging Datasets

Combining datasets is essential when integrating data from different sources.

- **Merging**: Joins datasets based on a common column.
- **Concatenation**: Stacks datasets along rows or columns.

Example:

```
import pandas as pd

# Merge example
data1 = pd.DataFrame({'ID': [1, 2], 'Name': ['Alice', 'Bob']})
data2 = pd.DataFrame({'ID': [1, 2], 'Score': [85, 90]})

merged = pd.merge(data1, data2, on='ID')
print(merged)
```

Reshaping and Pivoting

Reshaping reorganizes the structure of datasets.

- **Pivot**: Converts rows into columns or vice versa.
- **Melt**: Converts wide-format data into long-format.

Example:

```
# Pivot Table
pivot = data.pivot(index='City', columns='Year',
values='Value')
print(pivot)
```

---

8. Tools for Data Wrangling

Data Cleaning and Preparation

Data cleaning ensures data quality by handling errors, duplicates, and inconsistencies.

- **Handling Missing Data**: Replace, drop, or interpolate missing values.
- **Normalization**: Adjust data to a common scale.

Handling Missing Data

```
import pandas as pd

data = pd.DataFrame({'A': [1, None, 3], 'B': [4, 5, None]})
print(data.fillna(0))
```

Data Transformation

```
# Normalization
normalized = (data - data.min()) / (data.max() - data.min())
print(normalized)
```

String Manipulation String operations allow cleaning and formatting of text data.

Example:

```
# String operations
data['Name'] = data['Name'].str.upper()
print(data)
```

---

9. Summary

Data wrangling is a critical skill for data scientists. It ensures that data is clean, structured, and ready for analysis. With Python's powerful libraries,

these tasks can be performed efficiently.