Pandas and PySpark: Data Handling and Analysis

1. Pandas: Reading Data from Various Sources

Overview

Pandas supports importing and exporting data from multiple file formats and databases, making it highly versatile for data manipulation.

Reading Data

- **CSV Files**:

  ```
  import pandas as pd
  df = pd.read_csv('data.csv')
  ```

  Options:

  - `sep`: Define a custom delimiter (e.g., `sep=';'` for semicolon-separated files).
  - `header`: Specify row number for column names.

- **Excel Files**:

  ```
  df = pd.read_excel('data.xlsx', sheet_name='Sheet1')
  ```

- **JSON Files**:

  ```
  df = pd.read_json('data.json')
  ```

- **SQL Databases**:

  ```
  from sqlalchemy import create_engine
  engine = create_engine('sqlite:///database.db')
  df = pd.read_sql('SELECT * FROM table_name', engine)
  ```

- **Other Formats**:

  - HTML: `pd.read_html('file.html')`.
  - Parquet: `pd.read_parquet('file.parquet')`.

2. Key Pandas Data Structures

1. Series

- **Definition**: A one-dimensional labeled array capable of holding data of any type.

- **Example**:

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

2. DataFrame

- **Definition**: A two-dimensional, size-mutable, tabular data structure with labeled axes (rows and columns).
- **Example**:

```
data = {'Name': ['Alice', 'Bob'], 'Age': [25, 30]}
df = pd.DataFrame(data)
```

3. Arrays

- **Definition**: NumPy arrays are often used with Pandas for efficient numerical computations.
- **Conversion Example**:

```
import numpy as np
df_array = df.to_numpy()
```

4. Vectors

- Represented as Series or a single column in a DataFrame.
- **Example**:

```
vector = df['Age']  # Extracting a column as a vector.
```

3. PySpark: Introduction

What is PySpark?

- PySpark is the Python API for **Apache Spark**, a distributed computing system designed for big data processing.
- Combines the power of Spark's distributed system with Python's simplicity.

Core Features of PySpark

- **Distributed Computing**: Handles large datasets by distributing computation across clusters.
- **In-Memory Processing**: Provides faster data processing by caching data in memory.
- **Integration**: Works with Hadoop, HDFS, and various data sources (e.g., Parquet, JSON).
- **Machine Learning**: Offers MLlib for scalable machine learning tasks.

Key Components

- **SparkContext**: Entry point for Spark functionality.

```
from pyspark import SparkContext
sc = SparkContext('local', 'example')
```

- **DataFrame API**: High-level abstraction for working with structured data.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('example').getOrCreate()
df = spark.read.csv('data.csv', header=True, inferSchema=True)
df.show()
```

- **RDD (Resilient Distributed Dataset)**: Low-level distributed data abstraction for transformation and action operations.

Pandas vs. PySpark

Aspect	Pandas	PySpark
Use Case	Small to medium datasets.	Large datasets, distributed.
Performance	Single machine, slower.	Cluster-based, highly scalable.
Ease of Use	Simple, Pythonic syntax.	More setup required.

Example PySpark DataFrame Operations

- **Reading Data**:

```
df = spark.read.csv('data.csv', header=True, inferSchema=True)
```

- **Basic Queries**:

```
df.filter(df['Age'] > 25).select('Name').show()
```

- **Aggregation**:

```
df.groupBy('Department').agg({'Salary': 'avg'}).show()
```

4. Summary of Key Concepts

Concept	Definition	Example
Pandas Series	1D labeled array.	`pd.Series([1, 2, 3])`
Pandas DataFrame	2D labeled data structure.	`pd.DataFrame({'A': [1, 2]})`
PySpark	Python API for distributed data	`spark.read.csv('data.csv')`

Concept	Definition	Example
	processing.	
Pandas Use Case	Small datasets, single machine.	Exploratory data analysis (EDA).
PySpark Use Case	Big data, distributed environments.	ETL processes, machine learning.

Conclusion

- **Pandas** excels in single-machine, small-scale data analysis with intuitive and flexible tools.
- **PySpark** is ideal for big data processing with its distributed framework and scalability.
- Together, they form a powerful combination for handling a wide range of data analysis tasks.