Revision Notes on Pandas Dataframes and Series

In this class, we delved into the fundamental aspects of using Pandas for data manipulation and analysis. Here's a comprehensive overview of the topics covered:

Introduction to Pandas

Pandas is a powerful data manipulation and analysis library for Python, built on top of Numpy. It provides data structures like `DataFrame` and `Series` which are essential for handling structured data efficiently.

Installation

To install Pandas, you can use pip in the command line:

```
pip install pandas
```

Importing Pandas

To use Pandas in your Python environment, you need to import it using:

```
import pandas as pd
```

Understanding DataFrames and Series

DataFrame

A **DataFrame** is a two-dimensional, size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).

- **Structure**: It is akin to a table in a SQL database or a data sheet in Excel.

Series

A **Series** is a one-dimensional labeled array capable of holding any data type.

- Acts like a single column of a DataFrame.
- **Creation**: You can create a Series from a list, dictionary, or scalar value.

Basic Data Operations

Loading Data

Reading CSV Data

You can load a CSV file into a DataFrame using:

```
df = pd.read_csv('file.csv')
```

This function reads the contents of a CSV file into a DataFrame, where each column in the CSV becomes a column in the DataFrame .

Inspecting Data

- **Head/Tail**: Use `df.head(n)` and `df.tail(n)` to view the first and last `n` entries of your DataFrame.
- **Info**: Use `df.info()` to get a concise summary of the DataFrame, including the index dtype, column dtypes, and non-null counts .
- **Shape**: `df.shape` returns a tuple representing the dimensionality of the DataFrame (rows, columns).

Indexing and Selecting Data

- **Accessing columns**: You can access a column as a Series using `df['column_name']` .

Modifying Data

- **Adding Columns**: Add new columns by assigning operating results or values to a new column name, e.g., `df['new_column'] = values`.
- **Renaming Columns**: Use `df.rename(columns={"old_name": "new_name"})`.
- **Dropping Columns**: Utilize `df.drop('column_name', axis=1, inplace=True)` to remove a column【4:14†source】.

Sorting and Filtering

- **Sorting**: Use `df.sort_values(by='column')` to sort the DataFrame based on a specific column.
- **Filtering**: You apply boolean indexing to filter data, e.g., `df[df['column'] > value]`.

Advanced Concepts

Data Types and Null Values

- **Data Types**: Use `df.dtypes` to find the data type of each column. Object datatype typically indicates string values .

Resetting Index

- **Reset Index**: Use `df.reset_index(drop=True)` to reset the indices without creating an additional column for the old index .

Grouping and Aggregation

- Use `df.groupby('column').sum()` to aggregate data based on unique column values.

Practical Application

Working with Real Data

data to draw insights such as correlations between GDP and life expectancy .

Conclusion

Pandas is a versatile tool for manipulating and analyzing structured data. Its intuitive syntax and numerous features make it indispensable for data science tasks. Always refer to official documentation and community resources for more advanced functionalities and troubleshooting.