Revision Notes: Pandas Class

Introduction to Pandas

Pandas is a powerful data manipulation library in Python primarily used for data analysis. It is preferred for handling and analyzing data due to its ability to manage heterogeneous data formats, unlike NumPy, which deals with uniform types 【4:2†typed.md】. Pandas structures the data in `DataFrames`, akin to Excel spreadsheets or SQL tables, and `Series`, which are similar to columns in Excel 【4:8†typed.md】.

Installation & Setup

To get started with Pandas, you need to install it via pip if it isn't already available in your Python environment:

```
!pip install pandas
```

Once installed, Pandas can be imported usually with an alias:

```
import pandas as pd
```

In practice, it is common to see libraries such as Pandas and NumPy (imported as `np`) used together 【4:0†transcript.txt】.

Working with DataFrames

Reading Data

A common first step in data analysis is reading data from a CSV file. The Pandas `read_csv` function is used to load datasets into a DataFrame. For example, to read in a `mckinsey.csv` file:

Exploring Data

Once loaded, you can explore the basic structure of a DataFrame using several methods:

- `df.head()` displays the first few rows (default is 5)【4:18†typed.md】.
- `df.info()` provides a summary of the DataFrame, including the number of entries, and the data type of each column【4:18†typed.md】.
- `df.shape` returns the dimensions of the DataFrame, i.e., the number of rows and columns【4:13†typed.md】.

DataFrame vs. Series

A DataFrame is a two-dimensional structure containing multiple series. Each series in a DataFrame is essentially a column. You can think of a DataFrame as a dictionary of series that share the same index【4:16†transcript.txt】【4:18†typed.md】.

Example:

- **DataFrame**: Represents a full table with multiple columns and rows.
- **Series**: Represents a single column in a DataFrame.

```
# Accessing a column returns a Series
country_series = df['country']
```

Indexing and Selecting Data

Data can be accessed in Pandas using several techniques, and understanding the difference between implicit and explicit indexing is crucial:

- **Implicit Indexing** uses positions like lists in Python.
- **Explicit Indexing** uses named indices(if provided).

For instance, using `df['country']` would explicitly access the 'country' column【4:16†transcript.txt】【4:17†transcript.txt】.

1. **Adding a Column**: To add a new column, you could perform operations on an existing column or create a new series:

```
df['new_column'] = df['year'] + 7
```

2. **Renaming Columns**: Changing column names can make the DataFrame more readable or consistent with your analysis needs:

```
df.rename(columns={'old_name': 'new_name'}, inplace=True)
```

3. **Dropping Columns**: To remove columns that are unnecessary for analysis:

```
df.drop(columns=['column_to_drop'], inplace=True)
```

4. **Value Counts**: This function counts the number of occurrences of each unique value in a column:

```
df['country'].value_counts()
```

Understanding the Data

The dataset used for this class involved aspects like country, population, life expectancy, etc., aiming to analyze economic and demographic indicators for insights like the relation between GDP per capita and life expectancy 【4:2†typed.md】.

By leveraging the tools and methods demonstrated, practitioners can efficiently manipulate and investigate datasets, lay the groundwork for deeper analysis, or prepare data for machine learning pipelines.

Conclusion

Pandas is an indispensable library for data science with Python due to its flexibility and comprehensive feature set for manipulating structured data. Familiarity with its core concepts and operations

Students are encouraged to practice and explore further functionalities to deepen their understanding and proficiency in using Pandas for data manipulation【4:4†transcript.txt】【4:11†transcript.txt】.