Introduction to Pandas

Pandas is a powerful and flexible open-source data analysis and manipulation library for Python. It provides data structures and functions needed to work with structured data seamlessly.

1. Importing Pandas

Before you can use Pandas, you need to import it. Typically, it is imported with the alias pd.

```
import pandas as pd
```

---

2. Creating DataFrames

From a Dictionary

You can create a DataFrame from a dictionary where the keys are the column names and the values are lists of column values.

```
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['New York', 'Los Angeles', 'Chicago']
}
df = pd.DataFrame(data)
```

From a CSV File

You can also create a DataFrame by reading data from a CSV file.

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

---

3. DataFrame Basic Operations

Viewing Data

- **Head**: Display the first few rows of the DataFrame.

```
df.head()
```

- **Tail**: Display the last few rows of the DataFrame.

```
df.tail()
```

- **Info**: Get a concise summary of the DataFrame.

```
df.info()
```

- **Describe**: Get descriptive statistics.

```
df.describe()
```

Accessing Data

- **Access a Column**: You can access a column using the column name.

```
df['Name']
```

- **Access Multiple Columns**: You can access multiple columns by passing a list of column names.

```
df[['Name', 'Age']]
```

- **Access Rows by Index**: Use `iloc` to access rows by their integer index.

```
df.iloc[0]
```

- **Access Rows by Label**: Use `loc` to access rows by their labels.

```
df.loc[0]
```

---

4. Data Cleaning

Handling Missing Data

- **Check for Missing Values**: Check for missing values in the DataFrame.

```
df.isnull().sum()
```

- **Fill Missing Values**: Fill missing values with a specified value.

```
df.fillna(0)
```

- **Drop Missing Values**: Drop rows with missing values.

```
df.dropna()
```

Data Type Conversion

- **Convert Data Types**: Convert columns to different data types.

```
df['Age'] = df['Age'].astype(int)
```

---

5. Data Manipulation

Adding Columns

- **Add a New Column**: Add a new column by assigning a list or series.

```
df['Salary'] = [50000, 60000, 70000]
```

Removing Columns

- **Drop a Column**: Drop a column by its name.

```
df.drop('Salary', axis=1, inplace=True)
```

Filtering Data

- **Filter Rows**: Filter rows based on a condition.

```
df[df['Age'] > 30]
```

Grouping Data

- **Group By**: Group data by a specific column and perform an aggregate function.

```
df.groupby('City')['Age'].mean()
```

---

6. Merging and Joining

Concatenation

- **Concatenate DataFrames**: Concatenate two DataFrames along rows or columns.

```
pd.concat([df1, df2])
```

Merging

- **Merge DataFrames**: Merge two DataFrames based on a common column.

```
pd.merge(df1, df2, on='key_column')
```

Conclusion

Pandas is a comprehensive tool that provides flexible and powerful data manipulation capabilities. These notes cover the basics, but Pandas offers many more functionalities for handling complex data analysis tasks.

You can use this template to document your Pandas notebook, adding specific code examples and explanations relevant to the content of your notebook. If you have any specific sections or topics in the notebook that you want detailed notes for, please let me know!

Working with Pandas

In this tutorial, we will cover basic operations in Pandas including creating DataFrames, accessing data, filtering, adding and dropping columns, renaming columns, sorting, grouping, and saving data to a CSV file.

1. Importing the Pandas Library

First, we need to import the Pandas library, which is typically done with the alias `pd`.

```
import pandas as pd
```

2. Creating a DataFrame from a Dictionary

We can create a DataFrame by passing a dictionary where keys are column names and values are lists of column values.

```
data = {
    'Name': ['John', 'Emma', 'Michael', 'Sophia'],
    'Age': [25, 28, 32, 30],
    'City': ['New York', 'London', 'Paris', 'Tokyo']
}
df = pd.DataFrame(data)
```

Displaying the DataFrame

To display the DataFrame, we can simply print it.

```
print(df)
```

3. Accessing Data

Accessing Columns

Columns can be accessed using the column name.

```
print(df['Name'])  # Access using column name
print(df.Age)      # Access using attribute-style access
```

Accessing Rows

Rows can be accessed using the `loc` and `iloc` methods.

```
print(df.loc[0])   # Access by label
print(df.iloc[1])  # Access by index
```

4. Filtering Data

We can filter data based on a condition.

```
filtered_df = df[df['Age'] > 28]
print(filtered_df)
```

5. Adding a New Column

To add a new column, we can assign a list of values to a new column name.

```
df['Gender'] = ['Male', 'Female', 'Male', 'Female']
print(df)
```

6. Dropping a Column

We can drop a column using the `drop` method. Here, we drop the 'City' column.

```
df = df.drop('City', axis=1)
print(df)
```

7. Renaming Columns

To rename columns, we can use the `rename` method and pass a dictionary where keys are the old column names and values are the new column names.

```
df = df.rename(columns={'Name': 'Full Name'})
print(df)
```

8. Sorting Data

We can sort the DataFrame by a specific column using the `sort_values` method.

```
sorted_df = df.sort_values(by='Age', ascending=False)
print(sorted_df)
```

9. Grouping Data

Grouping data can be done using the `groupby` method followed by an aggregation function. Here, we group by 'Gender' and calculate the mean of numeric columns.

```
grouped_df = df.groupby('Gender').mean()
print(grouped_df)
```

10. Saving the DataFrame to a CSV File

Finally, we can save the DataFrame to a CSV file using the `to_csv` method.

```
df.to_csv('data.csv', index=False)
```

This concludes our basic operations with Pandas. Each operation is fundamental for data manipulation and analysis.