Pandas JSON

Pandas offers methods like `read_json()` and `to_json()` to work with JSON (JavaScript Object Notation) data.

JSON is a plain text document that follows a format similar to a JavaScript object. It consists of key-value pairs, where the keys are strings, and the values can be strings, numbers, booleans, arrays, or even other JSON objects.

Here's an example of a JSON:

```json { "Name": "Alice", "Age": 25, "City": "New York", "Skills": ["Python", "Data Analysis", "Machine Learning"] }

```
[
  {
    "name": "John",
    "age": 30,
    "city": "New York"
  },
  {
    "name": "Emily",
    "age": 28,
    "city": "San Francisco"
  },
  {
    "name": "David",
    "age": 35,
    "city": "Chicago"
  }
]
```

Let's name this JSON file `json_data.json`.

## Read JSON in Pandas

To read JSON data into a Pandas DataFrame, you can use the `read_json()` function.

Let's read the JSON file `json_data.json` we created before.

```
import pandas as pd

df = pd.read_json('json_output.json')

print(df)
```
```

read_json() Syntax

The syntax of `read_json()` in Pandas is:

```python df = pd.read_json(filepath_or_buffer, orient=None, typ='frame', numpy=False, precise_float=False,encoding=None,lines=False)

Here,

- **`filepath_or_buffer` (optional)**: specifies the path or URL to the JSON file or a file-like object containing the JSON data.
- **`orient` (optional)**: specifies the orientation of the JSON file. Common options include `'columns'`, `'index'`, `'split'`, etc.
- **`typ` (optional)**: indicates the type of expected output. Can be `'series'` or `'frame'` (default).
- **`precise_float` (optional)**: specifies whether to parse floats precisely.
- **`encoding` (optional)**: specifies the encoding to be used when reading the JSON file.
- **`lines` (optional)**: controls various aspects of the data reading process, especially for JSON files that contain multiple JSON objects (when set to `True`).

These are some commonly used arguments of the `read_json()` function. There are many other optional arguments that can be used with `read_json()`.

## Example: Read JSON

Let's suppose that we have a JSON file named `data.json` with the following contents:

```json [[1, "John", 25.12345], [2, "Jane", 30.98765432155], [3, "Alex", 28.56]]

Here, the JSON contains an array of arrays in the same line. So, we pass the required arguments to the `read_json()` method accordingly.

Now, let's load this JSON file into a DataFrame:

```python import pandas as pd

df = pd.read_json('data.json', orient = 'values', lines = False)

print(df)

Output:

0 1 2 0 1 John 25.123450 1 2 Jane 30.987654 2 3 Alex 28.560000

In this example, we read a JSON file containing an array of arrays using `read_json()`. We specified some arguments while reading the file to load the necessary data in appropriate format.

Here,
```

`orient = 'values'`: specifies that the JSON file contains an array of arrays `lines = False`: indicates that the JSON file does not have each row in a separate line To visualize the effect of orient and lines arguments, let's take a JSON in a different format.

```JSON {"id": 1, "name": "John", "value": 25.12345} {"id": 2, "name": "Jane", "value": 30.98765432155} {"id": 3, "name": "Alex", "value": 28.56} Note that the above JSON is in the wrong format. We're using it only to demonstrate the use of specified arguments.

Now, let's read the above JSON from `data.json`.

```python import pandas as pd

df = pd.read_json('data.json', orient = 'records', lines = True)

print(df) Output

id name value 0 1 John 25.123450 1 2 Jane 30.987654 2 3 Alex 28.560000

Here,

`orient = 'records'`: specifies that the JSON file contains data in key-value pairs `lines = True`: indicates that the JSON file contains each row in a separate line

# Write JSON in Pandas

To write a Pandas DataFrame to a JSON file, you can use the `to_json()` function. For example,

```python
import pandas as pd

# create a dictionary
data = {'Name': ['John', 'Alice', 'Bob'],
        'Age': [25, 30, 35],
        'City': ['New York', 'London', 'Paris']}

# create a dataframe from the dictionary
df = pd.DataFrame(data)

# write dataframe to json file
df.to_json('json_output.json')
```

```python Output: {"Name":{"0":"John","1":"Alice","2":"Bob"},"Age":{"0":25,"1":30,"2":35},"City":{"0":"New York","1":"London","2":"Paris"}}

The above code snippet writes the `df` DataFrame to the JSON file `output.json`.

# to_json() Syntax

The syntax of `to_json()` in Pandas is: ```python df.to_json( path_or_buf, orient= 'columns', lines=False, compression='infer', index=True )

Here,
```

- **`path_or_buf` (optional)**: specifies the file path or buffer where the JSON string is written.
- **`orient` (optional)**: specifies the format of the JSON string. Common options include `'records'`, `'split'`, `'index'`, `'columns'`, etc.
- **`lines` (optional)**: specifies whether the resulting JSON string should be in a line-separated format (useful for large JSON files).
- **`compression` (optional)**: specifies the compression algorithm for file output (e.g., `'gzip'`, `'bz2'`, `'zip'`).
- **`index` (optional)**: specifies whether to include the DataFrame's index in the JSON string.

These are some commonly used arguments of the `to_json()` function. There are many other optional arguments that can be used with `to_json()`.

Example: Write JSON

```
import pandas as pd

# create a dictionary
data = {'Name': ['John', 'Alice', 'Bob'],
             'Age': [25, 30, 35],
             'City': ['New York', 'London', 'Paris']}

# create a dataframe from the dictionary
df = pd.DataFrame(data)

# write dataframe to json file
df.to_json('output5.json', orient = 'records', indent = 4)
```

Output

```json [ { "Name": "John", "Age": 25, "City": "New York" }, { "Name": "Alice", "Age": 30, "City": "London" }, { "Name": "Bob", "Age": 35, "City": "Paris" }]

In this example, we exported the DataFrame `df` to the `output.json` file.

Here,

- **`orient='records'`**: represents each row in the DataFrame as a JSON object.
- **`indent=4`**: sets the number of spaces used for indentation to 4.

**Note**: The above code will create a new file named `output.json` in the current directory (unless a different directory is specified in the file path).

If the file `output.json` already exists in the current directory, running this code will overwrite the existing file with the new contents of the DataFrame.

```
```
```