Pandas CSV

Pandas provides functions for both reading from and writing to CSV files.

CSV stands for **Comma-Separated Values**. It is a popular file format used for storing tabular data, where each row represents a record, and columns are separated by a delimiter (generally a comma).

For example, contents of a CSV file may look like,

Employee ID,First Name,Last Name,Department,Position,Salary

- **101,John,Doe,Marketing,Manager,50000**
- **102,Jane,Smith,Sales,Associate,35000**
- **103,Michael,Johnson,Finance,Analyst,45000**
- **104,Emily,Williams,HR,Coordinator,40000**

Read CSV Files

In Pandas, the `read_csv()` function allows us to read data from a CSV file into a **DataFrame**. It automatically detects commas and parses the data into appropriate columns.

Here's an example of reading a CSV file using Pandas:

```
import pandas as pd

# read csv file
df = pd.read_csv('data.csv', header = 0)

print(df)
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.0
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

The above code reads the contents of the `data.csv` file and creates a DataFrame named `df` containing the data from the CSV file.

Here, `header=0` sets the first row as the header of the dataframe.

The contents of the `data.csv` file are the same as the contents of the CSV file provided in the introduction section.

Note: The `data.csv` file should be present in the current directory for the above code to work. If it is in a different directory, you will need to provide the full path to the file.

For example, if the file `data.csv` is in the folder `csv_files`, the path `'./csv_files/data.csv'` should be specified as:

```python df = pd.read_csv('./csv_files/data.csv', header=0)

# read_csv() Syntax

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

```
df = pd.read_csv(
    filepath_or_buffer,
    sep=',',
    header=0,
    names=['col1', 'col2', 'col3'],
    index_col='col1',
    usecols=['col1', 'col3'],
    skiprows=[1, 3],
    nrows=100,
    skipinitialspace=True
)
```

Here,

- **`filepath_or_buffer`**: represents the path or buffer object containing the CSV data to be read.
- **`sep` (optional)**: specifies the delimiter used in the CSV file.
- **`header` (optional)**: indicates the row number to be used as the header or column names.
- **`names` (optional)**: a list of column names to assign to the DataFrame.
- **`index_col` (optional)**: specifies the column to be used as the index of the DataFrame.
- **`usecols` (optional)**: a list of columns to be read and included in the DataFrame.
- **`skiprows` (optional)**: used to skip specific rows while reading the CSV file.
- **`nrows` (optional)**: sets the maximum number of rows to be read from the CSV file.
- **`skipinitialspace` (optional)**: determines whether to skip any whitespace after the delimiter in each field.

These are some commonly used arguments of the `read_csv()` function. All of them are optional except `filepath_or_buffer`. There are many other optional arguments of `read_csv()`.
```

To learn more, please refer to the official documentation on `read_csv()`.

Example: `read_csv()` With Arguments

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

- **23, 'Hello', 45.6**
- **56, 'World', 78.9**
- **89, 'Foo', 12.3**
- **34, 'Bar', 56.7**

Now, let's load this CSV file into a DataFrame.

```
import pandas as pd

# read csv file with some arguments
df = pd.read_csv('data.csv', header = None, names = ['col1', 'col2',
'col3'], skiprows = 2)

print(df)
```

```
    col1  col2   col3
60   117   145  479.0
60   103   135  340.0
45   109   175  282.4
45   117   148  406.0
60   102   127  300.0
..   ...   ...    ...
60   105   140  290.8
60   110   145  300.0
60   115   145  310.2
75   120   150  320.4
75   125   150  330.4

[168 rows x 3 columns]
```

In this example, we read a CSV file using the `read_csv()` method. We specified some arguments while reading the file to load the necessary data in the appropriate format.

Here,

- **`header=None`**: indicates that the file doesn't have a header row.
- **`names=['col1', 'col2', 'col3']`**: assigns the column names as `'col1'`, `'col2'`, and `'col3'`.
- **`skiprows=2`**: skips the first two rows.

Write to CSV Files

We used `read_csv()` to read data from a CSV file into a DataFrame.

Pandas also provides the `to_csv()` function to write data from a DataFrame into a CSV file.

Let's see an example:

```
import pandas as pd

# Create a DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'Los Angeles', 'Chicago']}

df = pd.DataFrame(data)

# Write the DataFrame to a CSV file
df.to_csv('output.csv', index=False)
output_csv= df.to_csv('output.csv', index=False)
print(pd.read_csv('output.csv'))
```

```
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
2  Charlie   35      Chicago
```

Here, the above code writes the DataFrame `df` to the `output.csv` file. The `index=False` parameter is used to exclude the index labels from the CSV file.

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

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

to_csv() Syntax

The syntax of to_csv() in Pandas is:

```
df.to_csv(
    path_or_buf,
    sep=',',
    header=True,
    index=False,
    mode='w',
    encoding=None,
    quoting=None,
    line_terminator='\n',
)
```

Here,

- **`path_or_buf`**: represents the path or buffer object where the DataFrame will be saved as a CSV file.
- **`sep` (optional)**: specifies the delimiter to be used in the output CSV file.

- **`header` (optional)**: indicates whether to include the header row in the output CSV file.
- **`index` (optional)**: determines whether to include the index column in the output CSV file.
- **`mode` (optional)**: specifies the mode in which the output file will be opened.
- **`encoding` (optional)**: sets the character encoding to be used when writing the CSV file.
- **`quoting` (optional)**: determines the quoting behavior for fields that contain special characters.
- **`line_terminator` (optional)**: specifies the character sequence used to terminate lines in the output CSV file.

These are some commonly used arguments of the `to_csv()` function. All of them are optional except `path_or_buf`. There are many other optional arguments that can be used with `to_csv()`.

Example: to_csv() With Arguments

```
import pandas as pd

# create dataframe
data = {'Name': ['Tom', 'Nick', 'John', 'Tom'],
        'Age': [20, 21, 19, 18],
        'City': ['New York', 'London', 'Paris', 'Berlin']}
df = pd.DataFrame(data)

# write to csv file
df.to_csv('output1.csv', sep = ';', index = False, header = True)
```

In this example, we wrote a DataFrame to the CSV file `'output.csv'` using the `to_csv()` method. We used some arguments to write the necessary data to the file in the required format.

Here,

- **`sep=';'`**: specifies the delimiter as `';'`.
- **`index=False`**: instructs not to include the index column in the output file.
- **`header=True`**: instructs to include the column names as the header in the output file.

```

```