🐼 Most Used Pandas Methods with Key Parameters and 3 Examples Each

This document covers the most frequently used Pandas methods, including a clear description, important parameters, and three practical examples for each.

1. read_csv()

Description: Reads data from a CSV file into a DataFrame.

Important Parameters:

- `filepath_or_buffer`: The path or URL to the CSV file
- `delimiter` / `sep`: String to use as the field separator
- `nrows`: Number of rows to read from the file
- `usecols`: Return a subset of the columns

Examples:

1. **Basic CSV reading:**

   python

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

2. **Custom delimiter:**

   python

   ```
   df = pd.read_csv('data.csv', delimiter=';')
   ```

3. **Limited rows and columns:**

   python

   ```
   df = pd.read_csv('data.csv', nrows=5, usecols=['Name', 'Age'])
   ```

---

2. head()

Description: Returns the first n rows of a DataFrame.

Important Parameters:

- `n`: Number of rows to return from the top. Default is 5

Examples:

1. **Default first 5 rows:**

```python
df.head()
```

2. **First 3 rows:**

```python
df.head(3)
```

3. **First 10 rows (explicit parameter):**

```python
df.head(n=10)
```

---

3. info()

Description: Provides a concise summary of a DataFrame including data types and memory usage.

Important Parameters:

- `verbose`: Whether to print the full summary. Default is None
- `memory_usage`: Whether to display memory usage. Default is 'deep'
- `show_counts`: Whether to show the non-null counts. Default is None

Examples:

1. **Basic info:**

```python
df.info()
```

2. **Detailed memory usage:**

```python
df.info(memory_usage='deep')
```

3. **Show counts for all columns:**

```python
df.info(show_counts=True)
```

---

4. describe()

Description: Generates descriptive statistics for numeric columns.

Important Parameters:

- `percentiles`: List of percentiles to include. Default is [.25, .5, .75]
- `include`: Data types to include ('all', 'number', 'object', etc.)
- `exclude`: Data types to exclude

Examples:

1. **Basic statistics:**

   python

   ```
   df.describe()
   ```

2. **Include all data types:**

   python

   ```
   df.describe(include='all')
   ```

3. **Custom percentiles:**

   python

   ```
   df.describe(percentiles=[.1, .5, .9])
   ```

---

5. fillna()

Description: Replaces missing values with a specified value.

Important Parameters:

- `value`: Scalar, dict, or Series used to fill NA
- `method`: 'ffill' or 'bfill' for forward/backward fill
- `inplace`: If True, modifies the original DataFrame

Examples:

1. **Fill all NaN with 0:**

   python

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

2. **Fill specific columns with different values:**

   python

   ```
   df.fillna({'Name': 'Unknown', 'Age': 0})
   ```

3. **Forward fill with in-place modification:**

```python
df.fillna(method='ffill', inplace=True)
```

---

6. dropna()

Description: Removes rows or columns containing missing values.

Important Parameters:

- `axis`: 0 for rows, 1 for columns. Default is 0
- `how`: 'any' or 'all'. Default is 'any'
- `subset`: Labels along other axis to consider
- `inplace`: If True, modifies the original DataFrame

Examples:

1. **Drop rows with any NaN:**

```python
df.dropna()
```

2. **Drop columns with all NaN:**

```python
df.dropna(axis=1, how='all')
```

3. **Drop rows where specific columns have NaN:**

```python
df.dropna(subset=['Name', 'Age'])
```

---

7. groupby()

Description: Groups DataFrame using a mapper or by a Series of columns.

Important Parameters:

- `by`: Used to determine the groups for grouping
- `axis`: 0 for rows, 1 for columns. Default is 0
- `as_index`: Return object with group labels as the index. Default is True
- `sort`: Sort group keys. Default is True

Examples:

1. **Group by single column:**

   python

   ```
   df.groupby('Category').sum()
   ```

2. **Group by multiple columns:**

   python

   ```
   df.groupby(['Category', 'Region']).mean()
   ```

3. **Group without sorting:**

   python

   ```
   df.groupby('Category', sort=False).count()
   ```

---

8. sort_values()

Description: Sorts DataFrame by the values along either axis.

Important Parameters:

- `by`: Name or list of names to sort by
- `axis`: 0 for rows, 1 for columns. Default is 0
- `ascending`: Sort ascending vs descending. Default is True
- `inplace`: If True, perform operation in-place

Examples:

1. **Sort by single column:**

   python

   ```
   df.sort_values('Age')
   ```

2. **Sort by multiple columns:**

   python

   ```
   df.sort_values(['Category', 'Price'], ascending=[True, False])
   ```

3. **Sort in descending order:**

   python

   ```
   df.sort_values('Salary', ascending=False)
   ```

---

9. drop()

Description: Drops specified labels from rows or columns.

Important Parameters:

- `labels`: Index or column labels to drop
- `axis`: 0 for rows, 1 for columns. Default is 0
- `index`: Alternative to specifying axis for rows
- `columns`: Alternative to specifying axis for columns
- `inplace`: If True, modifies the original DataFrame

Examples:

1. **Drop rows by index:**

   python

   ```
   df.drop([0, 2, 4])
   ```

2. **Drop columns:**

   python

   ```
   df.drop(['Name', 'Age'], axis=1)
   ```

3. **Drop columns using columns parameter:**

   python

   ```
   df.drop(columns=['Temp_Column'])
   ```

---

10. value_counts()

Description: Returns a Series containing counts of unique values.

Important Parameters:

- `normalize`: If True, return relative frequencies. Default is False
- `sort`: Sort by values. Default is True
- `ascending`: Sort in ascending order. Default is False
- `dropna`: Don't include counts of NaN. Default is True

Examples:

1. **Basic value counts:**

   python

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

2. **Normalized counts (percentages):**

   python

   ```
   df['Status'].value_counts(normalize=True)
   ```

3. **Include NaN counts:**

   python

   ```
   df['Grade'].value_counts(dropna=False)
   ```

---

11. to_csv()

Description: Writes a DataFrame to a CSV file.

Important Parameters:

- `path_or_buf`: The file path or object to write the CSV data
- `index`: Whether to write row names (index). Default is True
- `columns`: Subset of columns to write
- `sep`: Field delimiter for the output file

Examples:

1. **Basic CSV export:**

   python

   ```
   df.to_csv('output.csv')
   ```

2. **Export without index:**

   python

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

3. **Export specific columns with custom separator:**

   python

   ```
   df.to_csv('output.csv', columns=['Name', 'Age'], sep=';')
   ```

---

12. apply()

Description: Applies a function along an axis of the DataFrame.

Important Parameters:

- `func`: Function to apply to each column or row

- `axis`: 0 for rows, 1 for columns. Default is 0
- `raw`: Pass raw ndarray to function instead of Series. Default is False
- `args`: Positional arguments to pass to function

Examples:

1. **Apply function to each column:**

   python

   ```
   df.apply(lambda x: x.max() - x.min())
   ```

2. **Apply function to each row:**

   python

   ```
   df.apply(lambda x: x.sum(), axis=1)
   ```

3. **Apply custom function with arguments:**

   python

   ```
   df.apply(lambda x: x.clip(lower=0, upper=100))
   ```

---

13. merge()

Description: Merges DataFrame objects by performing a database-style join.

Important Parameters:

- `right`: DataFrame to merge with
- `how`: Type of merge ('left', 'right', 'outer', 'inner'). Default is 'inner'
- `on`: Column or index level names to join on
- `left_on` / `right_on`: Column names to join on for left/right DataFrame

Examples:

1. **Inner join on common column:**

   python

   ```
   df1.merge(df2, on='ID')
   ```

2. **Left join with different column names:**

   python

   ```
   df1.merge(df2, left_on='ID', right_on='UserID', how='left')
   ```

3. **Outer join on multiple columns:**

python

```
df1.merge(df2, on=['ID', 'Date'], how='outer')
```

---

14. isnull()

Description: Detects missing values in a DataFrame, returning a boolean mask.

Important Parameters:

- No key parameters (simple method)

Examples:

1. **Check for null values:**

   python

   ```
   df.isnull()
   ```

2. **Count null values per column:**

   python

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

3. **Check specific column for nulls:**

   python

   ```
   df['Age'].isnull()
   ```

---

15. loc[]

Description: Access a group of rows and columns by labels or a boolean array.

Important Parameters:

- `key`: Single label, list of labels, slice object, or boolean array
- Supports both row and column selection with `[row_indexer, column_indexer]`

Examples:

1. **Select rows by label:**

   python

   ```
   df.loc[0:2]  # Rows 0 to 2
   ```

2. **Select specific rows and columns:**

python

```
df.loc[0:2, ['Name', 'Age']]
```

3. **Boolean indexing:**

python

```
df.loc[df['Age'] > 25, 'Name']
```