

Read/Write and Inspection

Read/Write

<code>df = pd.read_excel('file.xlsx')</code>	Read file (<i>CSV, Excel, JSON, HTML, SQL</i>)
<code>df.to_csv('file.csv')</code>	Write file

Inspection

<code>df.head()/.tail()</code>	View first/last rows
<code>df.shape</code>	Gives dimensions
<code>df.dtypes</code>	Shows types in each column (<i>int, str etc</i>)
<code>df.info</code>	Lists: (<i>range of index, list of all columns, no. non null, data types, memory usage</i>)
<code>df.describe</code>	Lists: (<i>count, mean, std deviation, min, 25%, 50%, 75%, max</i>)

Filter

Functionality

<code>df[df['column'] vs df['column']</code>	1st filters dataframe, 2nd creates list of booleans
<code>df[df[x] > y & df[df[a] > v</code>	& combines filters
<code>df[df.apply(lambda row: row['col1'] * 2 > row['col2'], axis=0)</code>	Filter by function. Axis=0 for columns, 1 for rows.
<code>df.reset_index(drop=True)</code>	Reset indexes for filter

Rows and Columns

<code>df[['column1', 'column2']]</code>	Filter by columns
<code>df.iloc[1:5]</code>	Filter by row (2nd - 6th)
<code>df[df['column'] > value</code>	Filters rows based on boolean
<code>df[df['column'].isin([value1, value2])</code>	Filter rows based on list of values
<code>df.query('age > @min_age')</code>	Filter rows based on query string, @ points to variable 'and' to combine
<code>df.loc[df['column'] > value, 'column_name']</code>	Filters rows and columns

Dropping and replacing

<code>df.drop(columns=['column1', 'column2'])</code> or <code>df.drop(['unnecessary_column'], axis=1, inplace=True)</code>	Drop columns
<code>df.drop_duplicates(subset=['column'])</code>	Drop duplicate rows from specific columns
<code>df.dropna()</code> vs <code>.notna()</code>	Removes rows with missing values/non-missing values
<code>df.mask/.where(df['column'] > value)</code>	Replaces rows not meeting/meeting condition with NaN



Filter (cont)

String

<code>df[df['column'].str.contains('substring')]</code>	Filter where string column contains substring
<code>df[df['column'].str.match(r'abc\$')]</code>	<p><code>^abc</code>: Start with 'abc'; <code>abc\$</code>: End with 'abc'; <code>a b</code>: 'a' or 'b'; <code>abc+</code>: 'abc' followed by 1+ 'c's; <code>abc*</code>: 'ab' followed by 0+ 'c's; <code>abc?</code>: 'ab' followed by 0 or 1 'c'; <code>[abc]</code>: Any one of 'a', 'b', or 'c'</p>

Data Manipulation

Editing data

<code>df['new_column'] = new_values</code>	Assign/create new values for column
<code>df['column'] *= value</code>	Multiply (<code>/=</code> , <code>-=</code> , <code>+=</code>) each entry in a column by a value.
<code>df.fillna(value or method)</code>	Fill Nan values
<code>df['column'].replace(to_replace, value)</code>	Replace values
<code>df['column'].rolling(window=7).mean()</code>	Rolling aggregate
<code>df.update(other_df)</code>	Updates values from other df
<code>df['column'].astype(dtype)</code>	Converts data type
<code>np.where(df['column'] > value, 'Value =True', 'Value =False')</code>	Create array based on new conditions
<code>df['column'].apply(lambda/function name)</code>	Apply functions to selection
<code>df.rename(columns={'old_name': 'new_name'})</code>	Rename columns

Combining data

<code>pd.merge(df1, df2, on='common column', how='inner')</code>	Merges two dfs based on a common column.
<code>pd.merge(df1, df2, left_index=True, right_index=True, how='outer')</code>	'inner' requires both df to have all columns filled vs 'outer', 'left' vs 'right' to filter for columns of either df
<code>pd.concat([df1, df2])</code>	Concatenate (combines all values)
<code>df['column'].map(mapping_dict)</code>	Create a dictionary from a list to map keys and values to each other



Group by

Pandas Group By
Multiple Aggregate Functions

```
df.groupby('column_to_group').agg({'column1': agg_func1, 'column2': agg_func2})
```

```
df.groupby('Day').agg({
    'Rain': pd.Series.sum,
    'Temp': pd.Series.mean
})
```

Index	Day	Rain	Temp
0	Monday	2	65
1	Monday	3.2	68
2	Monday	1	67
3	Tuesday	8.5	62
4	Tuesday	0	78
5	Tuesday	1.2	59

Index	Day	Rain	Temp
0	Monday	6.2	64.6
1	Tuesday	9.7	63.6

```
grouped = df.groupby(['Store', 'Product']).agg({'Total_Sales': ('Sales', 'sum'), 'Average_Cost': ('Cost', 'mean'), 'Sales_Count': ('Sales', 'count')})
```

```
filtered_grouped = grouped[grouped['Total_Sales'] > 500]
```

Aggregations: sum; mean; median; min; max; count; size; std (standard deviation); var (variance); first; last; product; unique(number of unique values)

Pivot

Pivot

```
df.pivot(index='foo', columns='baz', values='bar')
```

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

Stacked

	bar	A	B	C
foo				
one	1	2	3	
two	4	5	6	

Record

```
pivoted_df = df.pivot(index='Date', columns='Variable', values='Value')
```

```
pivot_table_df = df.pivot_table(index='Date', columns='Variable', values='Value', aggfunc='sum')
```

values (optional): Columns whose data will be aggregated.

index: Columns used as index.

columns (optional): Columns to pivot into new DataFrame's columns.

aggfunc: Aggregation function for values.

pivot_table: aggregates duplicates

Melt

Melt

```
df3.melt(id_vars=['first', 'last'])
```

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

```
melted_df = pd.melt(df, id_vars=['id_column'], var_name='variable_name', value_name='value')
```

id_vars (optional): Columns to keep unchanged. Otherwise default melted

value_vars (optional): Columns to melt.

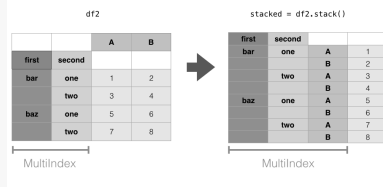
var_name (optional): Melted column name. Default is 'variable'.

value_name (optional): Melted values column name. Default is 'value'.

col_level (optional): Multi-index level

Stack/unstack

Stack



```
stacked_df = df.stack( level=-1, dropna =True)
```

Level: The level(s) of the column labels you want to stack. The default is the last level.

Dropna: Whether to drop rows in the resulting DataFrame with missing values. Default is True.



By Reszi
cheatography.com/reszi/

Published 28th January, 2024.
Last updated 29th January, 2024.
Page 3 of 5.

Sponsored by CrosswordCheats.com
Learn to solve cryptic crosswords!
<http://crosswordcheats.com>

Time Operations

<code>pd.to_datetime(df['column'])</code>	Converts a column to datetime format
<code>pd.Timestamp.now()</code>	Returns the current date and time
<code>df['datetime_column'].dt.date/.time</code>	Extracts the date/time from a datetime column
<code>datetime.datetime.strptime(date_string, format)</code>	Useful for strings with words and dates, or including hours/minutes, or multiple dates types in one column (requires try except function)

Open pyxl

<code>openpyxl.load_workbook(filename)</code> <code>workbook.save(filename)</code>	Read/Write
<code>workbook[sheetname]</code> or <code>workbook.active</code>	Sheet selection

Cell Formats

<code>cell.font = Font(size=12)</code>	Font
<code>cell.number_format = '0.00%'</code>	Cell formats
<code>Alignment(horizontal="center");</code> <code>Border(left=Side(border_style="thin", color="000000"));</code> <code>PatternFill(pattern="solid", fg_color="D9DDDD")</code>	More cell options
<code>Comment('Text', 'Author')</code>	Cell Comments, can change comment.width/height too.
<code>sheet.unmerge/merge_cells('A1:D1')</code>	Merge/Unmerge
<code>cell.hyperlink = 'http://www.example.com'</code>	Hyperlinks
<code>ColorScaleRule(start_type="min", start_color="FFFFFF", end_type="max", end_color="FF0000")</code> <code>ws.conditional_formatting.add('A1:A9', rule)</code>	Conditional Formatting
<code>DataValidation(type="list", formula1="Item1,Item2,Item3", showDropDown=True)</code> <code>ws.add_data_validation(dv)</code>	Data validation

Charts

<code>chart = BarChart(),</code> <code>chart.style = 13,</code> <code>sheet.add_chart(chart)</code>	Create, style, then add chart to sheet
<code>Image('path/to/image'), sheet.add_image(img, 'A1')</code>	Add images, better for matplotlib or other libraries
<code>cell.value = '=SUM(A1:A10)'</code>	Write formulas

