Search the docs ...

pandas.read_excel

pandas.**read_excel**(*io, sheet_name=0, header=0, names=None, index_col=None, usecols=None, squeeze=False, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skiprows=None, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, parse_dates=False, date_parser=None, thousands=None, comment=None, skipfooter=0, convert_float=True, mangle_dupe_cols=True, storage_options=None*) [source]

Read an Excel file into a pandas DataFrame.

Supports *xls*, *xlsx*, *xlsm*, *xlsb*, *odf*, *ods* and *odt* file extensions read from a local filesystem or URL. Supports an option to read a single sheet or a list of sheets.

Parameters: **io** : *str, bytes, ExcelFile, xlrd.Book, path object, or file-like object*

Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, and file. For file URLs, a host is expected. A local file could be: `file://localhost/path/to/table.xlsx`.

If you want to pass in a path object, pandas accepts any `os.PathLike`.

By file-like object, we refer to objects with a `read()` method, such as a file handle (e.g. via builtin `open` function) or `StringIO`.

sheet_name : *str, int, list, or None, default 0*

Strings are used for sheet names. Integers are used in zero-indexed sheet positions. Lists of strings/integers are used to request multiple sheets. Specify None to get all sheets.

Available cases:

- Defaults to `0`: 1st sheet as a *DataFrame*
- `1`: 2nd sheet as a *DataFrame*
- `"Sheet1"`: Load sheet with name "Sheet1"
- `[0, 1, "Sheet5"]`: Load first, second and sheet named "Sheet5" as a dict of *DataFrame*
- None: All sheets.

header : *int, list of int, default 0*

Row (0-indexed) to use for the column labels of the parsed DataFrame. If a list of integers is passed those row positions will be combined into a `MultiIndex`. Use None if there is no header.

names : *array-like, default None*

List of column names to use. If file contains no header row, then you should explicitly pass header=None.

index_col : *int, list of int, default None*

Column (0-indexed) to use as the row labels of the DataFrame. Pass None if there is no such column. If a list is passed, those columns will be combined into a `MultiIndex`. If a subset of data is selected with `usecols`, index_col is based on the subset.

usecols : *int, str, list-like, or callable default None*

- If None, then parse all columns.
- If str, then indicates comma separated list of Excel column letters and column ranges (e.g. "A:E" or "A,C,E:F"). Ranges are inclusive of both sides.
- If list of int, then indicates list of column numbers to be parsed.
- If list of string, then indicates list of column names to be parsed.
  New in version 0.24.0.
- If callable, then evaluate each column name against it and parse the column if the callable returns `True`.

Returns a subset of the columns according to behavior above.

New in version 0.24.0.

squeeze : *bool, default False*

If the parsed data only contains one column then return a Series.

dtype : *Type name or dict of column -> type, default None*

Data type for data or columns. E.g. {'a': np.float64, 'b': np.int32} Use *object* to preserve data as stored in Excel and not interpret dtype. If converters are specified, they will be applied INSTEAD of dtype conversion.

engine : *str, default None*

If io is not a buffer or path, this must be set to identify io. Supported engines: "xlrd", "openpyxl", "odf", "pyxlsb". Engine compatibility :

- "xlrd" supports old-style Excel files (.xls).
- "openpyxl" supports newer Excel file formats.
- "odf" supports OpenDocument file formats (.odf, .ods, .odt).
- "pyxlsb" supports Binary Excel files.

Changed in version 1.2.0: The engine xlrd now only supports old-style `.xls` files. When `engine=None`, the following logic will be used to determine the engine:

- If `path_or_buffer` is an OpenDocument format (.odf, .ods, .odt), then odf will be used.
- Otherwise if `path_or_buffer` is an xls format, `xlrd` will be used.
- Otherwise if openpyxl is installed, then `openpyxl` will be used.
- Otherwise if `xlrd >= 2.0` is installed, a `ValueError` will be raised.

Returns: **DataFrame or dict of DataFrames**

DataFrame from the passed in Excel file. See notes in sheet_name argument for more information on when a dict of DataFrames is returned.

See also

`DataFrame.to_excel`
Write DataFrame to an Excel file.

`DataFrame.to_csv`
Write DataFrame to a comma-separated values (csv) file.

`read_csv`
Read a comma-separated values (csv) file into DataFrame.

`read_fwf`
Read a table of fixed-width formatted lines into DataFrame.

Examples

The file can be read using the file name as string or an open file object:

```
>>> pd.read_excel('tmp.xlsx', index_col=0)
       Name  Value
0   string1      1
1   string2      2
2  #Comment      3
```

```
>>> pd.read_excel(open('tmp.xlsx', 'rb'),
...               sheet_name='Sheet3')
   Unnamed: 0      Name  Value
0           0   string1      1
1           1   string2      2
2           2  #Comment      3
```


Index and header can be specified via the *index_col* and *header* arguments

```
>>> pd.read_excel('tmp.xlsx', index_col=None, header=None)
     0         1      2
0  NaN      Name  Value
1  0.0   string1      1
2  1.0   string2      2
3  2.0  #Comment      3
```

Column types are inferred but can be explicitly specified

```
>>> pd.read_excel('tmp.xlsx', index_col=0,
...               dtype={'Name': str, 'Value': float})
       Name  Value
0   string1    1.0
1   string2    2.0
2  #Comment    3.0
```

True, False, and NA values, and thousands separators have defaults, but can be explicitly specified, too. Supply the values you would like as strings or lists of strings!

```
>>> pd.read_excel('tmp.xlsx', index_col=0,
...               na_values=['string1', 'string2'])
       Name  Value
0       NaN      1
1       NaN      2
2  #Comment      3
```

Comment lines in the excel input file can be skipped using the *comment* kwarg

```
>>> pd.read_excel('tmp.xlsx', index_col=0, comment='#')
      Name  Value
0  string1    1.0
1  string2    2.0
2     None    NaN
```

<< pandas.read_clipboard

pandas.ExcelFile.parse >>