
pandas: powerful Python data analysis toolkit

Release 1.0.5

Wes McKinney and the Pandas Development Team

Jun 17, 2020

CONTENTS

1	Getting started	3
1.1	Installation	3
1.2	Intro to pandas	3
1.3	Coming from	5
1.4	Community tutorials	5
1.4.1	Installation	5
1.4.2	Package overview	10
1.4.3	10 minutes to pandas	13
1.4.4	Getting started tutorials	37
1.4.5	Essential basic functionality	81
1.4.6	Intro to data structures	149
1.4.7	Comparison with other tools	171
1.4.8	Tutorials	222
2	User Guide	225
2.1	IO tools (text, CSV, HDF5, . . .)	225
2.1.1	CSV & text files	226
2.1.2	JSON	261
2.1.3	HTML	275
2.1.4	Excel files	284
2.1.5	OpenDocument Spreadsheets	290
2.1.6	Binary Excel (.xlsb) files	291
2.1.7	Clipboard	291
2.1.8	Pickling	292
2.1.9	msgpack	295
2.1.10	HDF5 (PyTables)	295
2.1.11	Feather	322
2.1.12	Parquet	323
2.1.13	ORC	326
2.1.14	SQL queries	326
2.1.15	Google BigQuery	334
2.1.16	Stata format	334
2.1.17	SAS formats	337
2.1.18	SPSS formats	337
2.1.19	Other file formats	338
2.1.20	Performance considerations	338
2.2	Indexing and selecting data	341
2.2.1	Different choices for indexing	342
2.2.2	Basics	342
2.2.3	Attribute access	345

2.2.4	Slicing ranges	347
2.2.5	Selection by label	348
2.2.6	Selection by position	352
2.2.7	Selection by callable	355
2.2.8	IX indexer is deprecated	357
2.2.9	Indexing with list with missing labels is deprecated	358
2.2.10	Selecting random samples	360
2.2.11	Setting with enlargement	362
2.2.12	Fast scalar value getting and setting	363
2.2.13	Boolean indexing	364
2.2.14	Indexing with <code>isin</code>	366
2.2.15	The <code>where()</code> Method and Masking	368
2.2.16	The <code>query()</code> Method	372
2.2.17	Duplicate data	382
2.2.18	Dictionary-like <code>get()</code> method	385
2.2.19	The <code>lookup()</code> method	385
2.2.20	Index objects	385
2.2.21	Set / reset index	389
2.2.22	Returning a view versus a copy	391
2.3	<code>MultiIndex</code> / advanced indexing	394
2.3.1	Hierarchical indexing (<code>MultiIndex</code>)	395
2.3.2	Advanced indexing with hierarchical index	401
2.3.3	Sorting a <code>MultiIndex</code>	411
2.3.4	Take methods	414
2.3.5	Index types	416
2.3.6	Miscellaneous indexing FAQ	426
2.4	Merge, join, and concatenate	430
2.4.1	Concatenating objects	430
2.4.2	Database-style <code>DataFrame</code> or named <code>Series</code> joining/merging	443
2.4.3	Timeseries friendly merging	461
2.5	Reshaping and pivot tables	464
2.5.1	Reshaping by pivoting <code>DataFrame</code> objects	464
2.5.2	Reshaping by stacking and unstacking	466
2.5.3	Reshaping by <code>Melt</code>	474
2.5.4	Combining with stats and <code>GroupBy</code>	475
2.5.5	Pivot tables	476
2.5.6	Cross tabulations	480
2.5.7	Tiling	482
2.5.8	Computing indicator / dummy variables	482
2.5.9	Factorizing values	486
2.5.10	Examples	486
2.5.11	Exploding a list-like column	489
2.6	Working with text data	491
2.6.1	Text Data Types	491
2.6.2	String Methods	493
2.6.3	Splitting and replacing strings	496
2.6.4	Concatenation	500
2.6.5	Indexing with <code>.str</code>	505
2.6.6	Extracting substrings	506
2.6.7	Testing for Strings that match or contain a pattern	510
2.6.8	Creating indicator variables	511
2.6.9	Method summary	511
2.7	Working with missing data	512
2.7.1	Values considered “missing”	513

2.7.2	Inserting missing data	516
2.7.3	Calculations with missing data	516
2.7.4	Sum/prod of empties/nans	518
2.7.5	NA values in GroupBy	518
2.7.6	Filling missing values: fillna	519
2.7.7	Filling with a PandasObject	520
2.7.8	Dropping axis labels with missing data: dropna	522
2.7.9	Interpolation	522
2.7.10	Replacing generic values	531
2.7.11	String/regular expression replacement	532
2.7.12	Numeric replacement	535
2.7.13	Experimental NA scalar to denote missing values	538
2.8	Categorical data	542
2.8.1	Object creation	542
2.8.2	CategoricalDtype	548
2.8.3	Description	549
2.8.4	Working with categories	549
2.8.5	Sorting and order	553
2.8.6	Comparisons	556
2.8.7	Operations	559
2.8.8	Data munging	560
2.8.9	Getting data in/out	568
2.8.10	Missing data	569
2.8.11	Differences to R's <i>factor</i>	570
2.8.12	Gotchas	570
2.9	Nullable integer data type	574
2.9.1	Construction	574
2.9.2	Operations	576
2.9.3	Scalar NA Value	577
2.10	Nullable Boolean Data Type	578
2.10.1	Indexing with NA values	578
2.10.2	Kleene Logical Operations	578
2.11	Visualization	580
2.11.1	Basic plotting: plot	580
2.11.2	Other plots	583
2.11.3	Plotting with missing data	614
2.11.4	Plotting Tools	615
2.11.5	Plot Formatting	623
2.11.6	Plotting directly with matplotlib	647
2.12	Computational tools	648
2.12.1	Statistical functions	648
2.12.2	Window Functions	653
2.12.3	Aggregation	668
2.12.4	Expanding windows	672
2.12.5	Exponentially weighted windows	675
2.13	Group By: split-apply-combine	678
2.13.1	Splitting an object into groups	679
2.13.2	Iterating through groups	686
2.13.3	Selecting a group	687
2.13.4	Aggregation	688
2.13.5	Transformation	694
2.13.6	Filtration	701
2.13.7	Dispatching to instance methods	702
2.13.8	Flexible apply	703

2.13.9	Other useful features	705
2.13.10	Examples	715
2.14	Time series / date functionality	718
2.14.1	Overview	720
2.14.2	Timestamps vs. Time Spans	721
2.14.3	Converting to timestamps	722
2.14.4	Generating ranges of timestamps	726
2.14.5	Timestamp limitations	730
2.14.6	Indexing	730
2.14.7	Time/date components	738
2.14.8	DateOffset objects	739
2.14.9	Time Series-Related Instance Methods	753
2.14.10	Resampling	755
2.14.11	Time span representation	763
2.14.12	Converting between representations	769
2.14.13	Representing out-of-bounds spans	771
2.14.14	Time zone handling	772
2.15	Time deltas	779
2.15.1	Parsing	779
2.15.2	Operations	781
2.15.3	Reductions	785
2.15.4	Frequency conversion	785
2.15.5	Attributes	788
2.15.6	TimedeltaIndex	789
2.15.7	Resampling	793
2.16	Styling	793
2.16.1	Building styles	793
2.16.2	Finer control: slicing	796
2.16.3	Finer Control: Display Values	797
2.16.4	Builtin styles	797
2.16.5	Sharing styles	799
2.16.6	Other Options	800
2.16.7	Fun stuff	803
2.16.8	Export to Excel	804
2.16.9	Extensibility	805
2.17	Options and settings	806
2.17.1	Overview	806
2.17.2	Getting and setting options	807
2.17.3	Setting startup options in Python/IPython environment	808
2.17.4	Frequently Used Options	808
2.17.5	Available options	815
2.17.6	Number formatting	816
2.17.7	Unicode formatting	817
2.17.8	Table schema display	818
2.18	Enhancing performance	818
2.18.1	Cython (writing C extensions for pandas)	819
2.18.2	Using Numba	824
2.18.3	Expression evaluation via <code>eval()</code>	826
2.19	Scaling to large datasets	834
2.19.1	Load less data	835
2.19.2	Use efficient datatypes	836
2.19.3	Use chunking	838
2.19.4	Use other libraries	839
2.20	Sparse data structures	843

2.20.1	SparseArray	845
2.20.2	SparseDtype	845
2.20.3	Sparse accessor	846
2.20.4	Sparse calculation	846
2.20.5	Migrating	847
2.20.6	Interaction with scipy.sparse	849
2.21	Frequently Asked Questions (FAQ)	852
2.21.1	DataFrame memory usage	852
2.21.2	Using if/truth statements with pandas	854
2.21.3	NaN, Integer NA values and NA type promotions	856
2.21.4	Differences with NumPy	858
2.21.5	Thread-safety	858
2.21.6	Byte-Ordering issues	858
2.22	Cookbook	859
2.22.1	Idioms	859
2.22.2	Selection	863
2.22.3	MultiIndexing	867
2.22.4	Missing data	871
2.22.5	Grouping	872
2.22.6	Timeseries	882
2.22.7	Merge	882
2.22.8	Plotting	884
2.22.9	Data In/Out	885
2.22.10	Computation	890
2.22.11	Timedeltas	891
2.22.12	Aliasing axis names	893
2.22.13	Creating example data	894
3	API reference	895
3.1	Input/output	895
3.1.1	Pickling	895
3.1.2	Flat file	896
3.1.3	Clipboard	907
3.1.4	Excel	907
3.1.5	JSON	913
3.1.6	HTML	919
3.1.7	HDFStore: PyTables (HDF5)	921
3.1.8	Feather	925
3.1.9	Parquet	926
3.1.10	ORC	926
3.1.11	SAS	927
3.1.12	SPSS	928
3.1.13	SQL	928
3.1.14	Google BigQuery	931
3.1.15	STATA	933
3.2	General functions	935
3.2.1	Data manipulations	935
3.2.2	Top-level missing data	965
3.2.3	Top-level conversions	971
3.2.4	Top-level dealing with datetimelike	973
3.2.5	Top-level dealing with intervals	982
3.2.6	Top-level evaluation	984
3.2.7	Hashing	986
3.2.8	Testing	987

3.3	Series	987
3.3.1	Constructor	987
3.3.2	Attributes	1210
3.3.3	Conversion	1211
3.3.4	Indexing, iteration	1212
3.3.5	Binary operator functions	1213
3.3.6	Function application, groupby & window	1214
3.3.7	Computations / descriptive stats	1215
3.3.8	Reindexing / selection / label manipulation	1216
3.3.9	Missing data handling	1217
3.3.10	Reshaping, sorting	1217
3.3.11	Combining / joining / merging	1217
3.3.12	Time series-related	1218
3.3.13	Accessors	1218
3.3.14	Plotting	1318
3.3.15	Serialization / IO / conversion	1359
3.4	DataFrame	1359
3.4.1	Constructor	1359
3.4.2	Attributes and underlying data	1677
3.4.3	Conversion	1678
3.4.4	Indexing, iteration	1678
3.4.5	Binary operator functions	1679
3.4.6	Function application, GroupBy & window	1680
3.4.7	Computations / descriptive stats	1680
3.4.8	Reindexing / selection / label manipulation	1681
3.4.9	Missing data handling	1682
3.4.10	Reshaping, sorting, transposing	1682
3.4.11	Combining / joining / merging	1683
3.4.12	Time series-related	1683
3.4.13	Metadata	1684
3.4.14	Plotting	1684
3.4.15	Sparse accessor	1730
3.4.16	Serialization / IO / conversion	1731
3.5	Pandas arrays	1732
3.5.1	pandas.array	1732
3.5.2	Datetime data	1736
3.5.3	Timedelta data	1756
3.5.4	Timespan data	1765
3.5.5	Period	1765
3.5.6	Interval data	1780
3.5.7	Nullable integer	1794
3.5.8	Categorical data	1799
3.5.9	Sparse data	1804
3.5.10	Text data	1806
3.5.11	Boolean data with missing values	1808
3.6	Panel	1810
3.7	Index objects	1810
3.7.1	Index	1810
3.7.2	Numeric Index	1866
3.7.3	CategoricalIndex	1870
3.7.4	IntervalIndex	1879
3.7.5	MultiIndex	1891
3.7.6	DatetimeIndex	1908
3.7.7	TimedeltaIndex	1935

3.7.8	PeriodIndex	1943
3.8	Date offsets	1950
3.8.1	DateOffset	1950
3.8.2	BusinessDay	1955
3.8.3	BusinessHour	1959
3.8.4	CustomBusinessDay	1963
3.8.5	CustomBusinessHour	1967
3.8.6	MonthOffset	1971
3.8.7	MonthEnd	1975
3.8.8	MonthBegin	1978
3.8.9	BusinessMonthEnd	1982
3.8.10	BusinessMonthBegin	1986
3.8.11	CustomBusinessMonthEnd	1989
3.8.12	CustomBusinessMonthBegin	1994
3.8.13	SemiMonthOffset	1998
3.8.14	SemiMonthEnd	2002
3.8.15	SemiMonthBegin	2006
3.8.16	Week	2010
3.8.17	WeekOfMonth	2013
3.8.18	LastWeekOfMonth	2017
3.8.19	QuarterOffset	2021
3.8.20	BQuarterEnd	2025
3.8.21	BQuarterBegin	2028
3.8.22	QuarterEnd	2032
3.8.23	QuarterBegin	2036
3.8.24	YearOffset	2039
3.8.25	BYearEnd	2043
3.8.26	BYearBegin	2046
3.8.27	YearEnd	2050
3.8.28	YearBegin	2054
3.8.29	FY5253	2057
3.8.30	FY5253Quarter	2062
3.8.31	Easter	2066
3.8.32	Tick	2070
3.8.33	Day	2074
3.8.34	Hour	2078
3.8.35	Minute	2082
3.8.36	Second	2086
3.8.37	Milli	2090
3.8.38	Micro	2094
3.8.39	Nano	2098
3.8.40	BDay	2102
3.8.41	BMonthEnd	2105
3.8.42	BMonthBegin	2108
3.8.43	CBMonthEnd	2111
3.8.44	CBMonthBegin	2115
3.8.45	CDay	2118
3.9	Frequencies	2122
3.9.1	pandas.tseries.frequencies.to_offset	2122
3.10	Window	2123
3.10.1	Standard moving window functions	2123
3.10.2	Standard expanding window functions	2141
3.10.3	Exponentially-weighted moving window functions	2154
3.10.4	Window Indexer	2156

3.11	GroupBy	2157
3.11.1	Indexing, iteration	2157
3.11.2	Function application	2159
3.11.3	Computations / descriptive stats	2161
3.12	Resampling	2203
3.12.1	Indexing, iteration	2203
3.12.2	Function application	2204
3.12.3	Upsampling	2208
3.12.4	Computations / descriptive stats	2220
3.13	Style	2225
3.13.1	Styler constructor	2225
3.13.2	Styler properties	2240
3.13.3	Style application	2240
3.13.4	Builtin styles	2240
3.13.5	Style export and import	2241
3.14	Plotting	2241
3.14.1	pandas.plotting.andrews_curves	2241
3.14.2	pandas.plotting.autocorrelation_plot	2242
3.14.3	pandas.plotting.bootstrap_plot	2242
3.14.4	pandas.plotting.boxplot	2243
3.14.5	pandas.plotting.deregister_matplotlib_converters	2250
3.14.6	pandas.plotting.lag_plot	2250
3.14.7	pandas.plotting.parallel_coordinates	2250
3.14.8	pandas.plotting.plot_params	2251
3.14.9	pandas.plotting.radviz	2251
3.14.10	pandas.plotting.register_matplotlib_converters	2252
3.14.11	pandas.plotting.scatter_matrix	2254
3.14.12	pandas.plotting.table	2254
3.15	General utility functions	2255
3.15.1	Working with options	2255
3.15.2	Testing functions	2269
3.15.3	Exceptions and warnings	2272
3.15.4	Data types related functionality	2275
3.16	Extensions	2302
3.16.1	pandas.api.extensions.register_extension_dtype	2303
3.16.2	pandas.api.extensions.register_dataframe_accessor	2303
3.16.3	pandas.api.extensions.register_series_accessor	2304
3.16.4	pandas.api.extensions.register_index_accessor	2306
3.16.5	pandas.api.extensions.ExtensionDtype	2307
3.16.6	pandas.api.extensions.ExtensionArray	2310
3.16.7	pandas.arrays.PandasArray	2323
3.16.8	pandas.api.indexers.check_array_indexer	2323
4	Development	2327
4.1	Contributing to pandas	2327
4.1.1	Where to start?	2328
4.1.2	Bug reports and enhancement requests	2329
4.1.3	Working with the code	2329
4.1.4	Contributing to the documentation	2333
4.1.5	Contributing to the code base	2351
4.1.6	Contributing your changes to <i>pandas</i>	2364
4.2	pandas code style guide	2367
4.2.1	Patterns	2367
4.2.2	String formatting	2367

4.3	Pandas Maintenance	2369
4.3.1	Roles	2369
4.3.2	Tasks	2369
4.3.3	Issue Triage	2369
4.3.4	Closing Issues	2370
4.3.5	Reviewing Pull Requests	2371
4.3.6	Cleaning up old Issues	2371
4.3.7	Cleaning up old Pull Requests	2371
4.3.8	Becoming a pandas maintainer	2371
4.4	Internals	2372
4.4.1	Indexing	2372
4.4.2	Subclassing pandas data structures	2374
4.5	Extending pandas	2374
4.5.1	Registering custom accessors	2374
4.5.2	Extension types	2375
4.5.3	Subclassing pandas data structures	2378
4.5.4	Plotting backends	2381
4.6	Developer	2381
4.6.1	Storing pandas DataFrame objects in Apache Parquet format	2381
4.7	Policies	2384
4.7.1	Version Policy	2384
4.7.2	Python Support	2385
4.8	Roadmap	2385
4.8.1	Extensibility	2385
4.8.2	String data type	2385
4.8.3	Apache Arrow interoperability	2386
4.8.4	Block manager rewrite	2386
4.8.5	Decoupling of indexing and internals	2386
4.8.6	Numba-accelerated operations	2386
4.8.7	Documentation improvements	2387
4.8.8	Package docstring validation	2387
4.8.9	Performance monitoring	2387
4.8.10	Roadmap Evolution	2387
4.9	Developer Meetings	2388
4.9.1	Minutes	2388
4.9.2	Calendar	2388
5	Release Notes	2389
5.1	Version 1.0	2389
5.1.1	What's new in 1.0.5 (June 17, 2020)	2389
5.1.2	What's new in 1.0.4 (May 28, 2020)	2390
5.1.3	What's new in 1.0.3 (March 17, 2020)	2391
5.1.4	What's new in 1.0.2 (March 12, 2020)	2392
5.1.5	What's new in 1.0.1 (February 5, 2020)	2395
5.1.6	What's new in 1.0.0 (January 29, 2020)	2398
5.2	Version 0.25	2437
5.2.1	What's new in 0.25.3 (October 31, 2019)	2437
5.2.2	What's new in 0.25.2 (October 15, 2019)	2437
5.2.3	What's new in 0.25.1 (August 21, 2019)	2438
5.2.4	What's new in 0.25.0 (July 18, 2019)	2441
5.3	Version 0.24	2479
5.3.1	Whats new in 0.24.2 (March 12, 2019)	2479
5.3.2	Whats new in 0.24.1 (February 3, 2019)	2482
5.3.3	What's new in 0.24.0 (January 25, 2019)	2483

5.4	Version 0.23	2540
5.4.1	What's new in 0.23.4 (August 3, 2018)	2540
5.4.2	What's new in 0.23.3 (July 7, 2018)	2541
5.4.3	What's new in 0.23.2 (July 5, 2018)	2542
5.4.4	What's new in 0.23.1 (June 12, 2018)	2545
5.4.5	What's new in 0.23.0 (May 15, 2018)	2549
5.5	Version 0.22	2599
5.5.1	v0.22.0 (December 29, 2017)	2599
5.6	Version 0.21	2604
5.6.1	v0.21.1 (December 12, 2017)	2604
5.6.2	v0.21.0 (October 27, 2017)	2609
5.7	Version 0.20	2641
5.7.1	v0.20.3 (July 7, 2017)	2641
5.7.2	v0.20.2 (June 4, 2017)	2644
5.7.3	v0.20.1 (May 5, 2017)	2648
5.8	Version 0.19	2696
5.8.1	v0.19.2 (December 24, 2016)	2696
5.8.2	v0.19.1 (November 3, 2016)	2699
5.8.3	v0.19.0 (October 2, 2016)	2702
5.9	Version 0.18	2747
5.9.1	v0.18.1 (May 3, 2016)	2747
5.9.2	v0.18.0 (March 13, 2016)	2766
5.10	Version 0.17	2801
5.10.1	v0.17.1 (November 21, 2015)	2801
5.10.2	v0.17.0 (October 9, 2015)	2808
5.11	Version 0.16	2838
5.11.1	v0.16.2 (June 12, 2015)	2838
5.11.2	v0.16.1 (May 11, 2015)	2843
5.11.3	v0.16.0 (March 22, 2015)	2856
5.12	Version 0.15	2874
5.12.1	v0.15.2 (December 12, 2014)	2874
5.12.2	v0.15.1 (November 9, 2014)	2881
5.12.3	v0.15.0 (October 18, 2014)	2888
5.13	Version 0.14	2920
5.13.1	v0.14.1 (July 11, 2014)	2920
5.13.2	v0.14.0 (May 31, 2014)	2927
5.14	Version 0.13	2957
5.14.1	v0.13.1 (February 3, 2014)	2957
5.14.2	v0.13.0 (January 3, 2014)	2968
5.15	Version 0.12	2998
5.15.1	v0.12.0 (July 24, 2013)	2998
5.16	Version 0.11	3010
5.16.1	v0.11.0 (April 22, 2013)	3010
5.17	Version 0.10	3020
5.17.1	v0.10.1 (January 22, 2013)	3020
5.17.2	v0.10.0 (December 17, 2012)	3026
5.18	Version 0.9	3038
5.18.1	v0.9.1 (November 14, 2012)	3038
5.18.2	v0.9.0 (October 7, 2012)	3043
5.19	Version 0.8	3045
5.19.1	v0.8.1 (July 22, 2012)	3045
5.19.2	v0.8.0 (June 29, 2012)	3046
5.20	Version 0.7	3053
5.20.1	v0.7.3 (April 12, 2012)	3053

5.20.2	v.0.7.2 (March 16, 2012)	3055
5.20.3	v.0.7.1 (February 29, 2012)	3056
5.20.4	v.0.7.0 (February 9, 2012)	3057
5.21	Version 0.6	3063
5.21.1	v.0.6.1 (December 13, 2011)	3063
5.21.2	v.0.6.0 (November 25, 2011)	3064
5.22	Version 0.5	3066
5.22.1	v.0.5.0 (October 24, 2011)	3066
5.23	Version 0.4	3068
5.23.1	v.0.4.1 through v0.4.3 (September 25 - October 9, 2011)	3068
Bibliography		3071
Python Module Index		3073

Date: Jun 17, 2020 **Version:** 1.0.5

Download documentation: [PDF Version](#) | [Zipped HTML](#)

Useful links: [Binary Installers](#) | [Source Repository](#) | [Issues & Ideas](#) | [Q&A Support](#) | [Mailing List](#)

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the [Python](#) programming language.

[To the getting started guides](#)

[To the user guide](#)

[To the reference guide](#)

[To the development guide](#)

GETTING STARTED

1.1 Installation

Before you can use pandas, you'll need to get it installed.

Pandas is part of the [Anaconda](#) distribution and can be installed with Anaconda or Miniconda:

```
conda install pandas
```

Pandas can be installed via pip from [PyPI](#).

```
pip install pandas
```

[Learn more](#)

1.2 Intro to pandas

[Straight to tutorial...](#)

When working with tabular data, such as data stored in spreadsheets or databases, Pandas is the right tool for you. Pandas will help you to explore, clean and process your data. In Pandas, a data table is called a *DataFrame*.

[To introduction tutorial](#)

[To user guide](#)

[Straight to tutorial...](#)

Pandas supports the integration with many file formats or data sources out of the box (csv, excel, sql, json, parquet,...). Importing data from each of these data sources is provided by function with the prefix `read_*`. Similarly, the `to_*` methods are used to store data.

[To introduction tutorial](#)

[To user guide](#)

[Straight to tutorial...](#)

Selecting or filtering specific rows and/or columns? Filtering the data on a condition? Methods for slicing, selecting, and extracting the data you need are available in Pandas.

To introduction tutorial

To user guide

Straight to tutorial...

Pandas provides plotting your data out of the box, using the power of Matplotlib. You can pick the plot type (scatter, bar, boxplot,...) corresponding to your data.

To introduction tutorial

To user guide

Straight to tutorial...

There is no need to loop over all rows of your data table to do calculations. Data manipulations on a column work elementwise. Adding a column to a *DataFrame* based on existing data in other columns is straightforward.

To introduction tutorial

To user guide

Straight to tutorial...

Basic statistics (mean, median, min, max, counts...) are easily calculable. These or custom aggregations can be applied on the entire data set, a sliding window of the data or grouped by categories. The latter is also known as the split-apply-combine approach.

To introduction tutorial

To user guide

Straight to tutorial...

Change the structure of your data table in multiple ways. You can *melt()* your data table from wide to long/tidy form or *pivot()* from long to wide format. With aggregations built-in, a pivot table is created with a single command.

To introduction tutorial

To user guide

Straight to tutorial...

Multiple tables can be concatenated both column wise as row wise and database-like join/merge operations are provided to combine multiple tables of data.

To introduction tutorial

To user guide

Straight to tutorial...

Pandas has great support for time series and has an extensive set of tools for working with dates, times, and time-indexed data.

To introduction tutorial

To user guide

Straight to tutorial...

Data sets do not only contain numerical data. Pandas provides a wide range of functions to cleaning textual data and extract useful information from it.

To introduction tutorial

To user guide

1.3 Coming from...

Currently working with other software for data manipulation in a tabular format? You're probably familiar to typical data operations and know *what* to do with your tabular data, but lacking the syntax to execute these operations. Get to know the pandas syntax by looking for equivalents from the software you already know:

Learn more

Learn more

Learn more

Learn more

1.4 Community tutorials

The community produces a wide variety of tutorials available online. Some of the material is enlisted in the community contributed *Tutorials*.

1.4.1 Installation

The easiest way to install pandas is to install it as part of the [Anaconda](#) distribution, a cross platform distribution for data analysis and scientific computing. This is the recommended installation method for most users.

Instructions for installing from source, [PyPI](#), [ActivePython](#), various Linux distributions, or a [development version](#) are also provided.

Python version support

Officially Python 3.6.1 and above, 3.7, and 3.8.

Installing pandas

Installing with Anaconda

Installing pandas and the rest of the [NumPy](#) and [SciPy](#) stack can be a little difficult for inexperienced users.

The simplest way to install not only pandas, but Python and the most popular packages that make up the [SciPy](#) stack ([IPython](#), [NumPy](#), [Matplotlib](#), ...) is with [Anaconda](#), a cross-platform (Linux, Mac OS X, Windows) Python distribution for data analytics and scientific computing.

After running the installer, the user will have access to pandas and the rest of the [SciPy](#) stack without needing to install anything else, and without needing to wait for any software to be compiled.

Installation instructions for [Anaconda](#) can be found [here](#).

A full list of the packages available as part of the [Anaconda](#) distribution can be found [here](#).

Another advantage to installing Anaconda is that you don't need admin rights to install it. Anaconda can install in the user's home directory, which makes it trivial to delete Anaconda if you decide (just delete that folder).

Installing with Miniconda

The previous section outlined how to get pandas installed as part of the [Anaconda](#) distribution. However this approach means you will install well over one hundred packages and involves downloading the installer which is a few hundred megabytes in size.

If you want to have more control on which packages, or have a limited internet bandwidth, then installing pandas with [Miniconda](#) may be a better solution.

[Conda](#) is the package manager that the [Anaconda](#) distribution is built upon. It is a package manager that is both cross-platform and language agnostic (it can play a similar role to a pip and virtualenv combination).

[Miniconda](#) allows you to create a minimal self contained Python installation, and then use the [Conda](#) command to install additional packages.

First you will need [Conda](#) to be installed and downloading and running the [Miniconda](#) will do this for you. The installer [can be found here](#)

The next step is to create a new conda environment. A conda environment is like a virtualenv that allows you to specify a specific version of Python and set of libraries. Run the following commands from a terminal window:

```
conda create -n name_of_my_env python
```

This will create a minimal environment with only Python installed in it. To put your self inside this environment run:

```
source activate name_of_my_env
```

On Windows the command is:

```
activate name_of_my_env
```

The final step required is to install pandas. This can be done with the following command:

```
conda install pandas
```

To install a specific pandas version:

```
conda install pandas=0.20.3
```

To install other packages, IPython for example:

```
conda install ipython
```

To install the full [Anaconda](#) distribution:

```
conda install anaconda
```

If you need packages that are available to pip but not conda, then install pip, and then use pip to install those packages:

```
conda install pip
pip install django
```

Installing from PyPI

pandas can be installed via pip from [PyPI](#).

```
pip install pandas
```

Installing with ActivePython

Installation instructions for [ActivePython](#) can be found [here](#). Versions 2.7, 3.5 and 3.6 include pandas.

Installing using your Linux distribution's package manager.

The commands in this table will install pandas for Python 3 from your distribution. To install pandas for Python 2, you may need to use the `python-pandas` package.

Distribution	Status	Download / Repository Link	Install method
Debian	stable	official Debian repository	<code>sudo apt-get install python3-pandas</code>
Debian & Ubuntu	unstable (latest packages)	NeuroDebian	<code>sudo apt-get install python3-pandas</code>
Ubuntu	stable	official Ubuntu repository	<code>sudo apt-get install python3-pandas</code>
OpenSuse	stable	OpenSuse Repository	<code>zypper in python3-pandas</code>
Fedora	stable	official Fedora repository	<code>dnf install python3-pandas</code>
Centos/RHEL	stable	EPEL repository	<code>yum install python3-pandas</code>

However, the packages in the linux package managers are often a few versions behind, so to get the newest version of pandas, it's recommended to install using the `pip` or `conda` methods described above.

Installing from source

See the [contributing guide](#) for complete instructions on building from the git source tree. Further, see [creating a development environment](#) if you wish to create a *pandas* development environment.

Running the test suite

pandas is equipped with an exhaustive set of unit tests, covering about 97% of the code base as of this writing. To run it on your machine to verify that everything is working (and that you have all of the dependencies, soft and hard, installed), make sure you have `pytest >= 5.0.1` and `Hypothesis >= 3.58`, then run:

```
>>> pd.test()
running: pytest --skip-slow --skip-network C:\Users\TP\Anaconda3\envs\py36\lib\site-
-packages\pandas
===== test session starts =====
platform win32 -- Python 3.6.2, pytest-3.6.0, py-1.4.34, pluggy-0.4.0
```

(continues on next page)

(continued from previous page)

```
rootdir: C:\Users\TP\Documents\Python\pandasdev\pandas, inifile: setup.cfg
collected 12145 items / 3 skipped

.....S.....
.....S.....
.....

===== 12130 passed, 12 skipped in 368.339 seconds =====
```

Dependencies

Package	Minimum supported version
setuptools	24.2.0
NumPy	1.13.3
python-dateutil	2.6.1
pytz	2017.2

Recommended dependencies

- [numexpr](#): for accelerating certain numerical operations. `numexpr` uses multiple cores as well as smart chunking and caching to achieve large speedups. If installed, must be Version 2.6.2 or higher.
- [bottleneck](#): for accelerating certain types of nan evaluations. `bottleneck` uses specialized cython routines to achieve large speedups. If installed, must be Version 1.2.1 or higher.

Note: You are highly encouraged to install these libraries, as they provide speed improvements, especially when working with large data sets.

Optional dependencies

Pandas has many optional dependencies that are only used for specific methods. For example, `pandas.read_hdf()` requires the `pytables` package, while `DataFrame.to_markdown()` requires the `tabulate` package. If the optional dependency is not installed, pandas will raise an `ImportError` when the method requiring that dependency is called.

Dependency	Minimum Version	Notes
BeautifulSoup4	4.6.0	HTML parser for <code>read_html</code> (see note)
Jinja2		Conditional formatting with <code>DataFrame.style</code>
PyQt4		Clipboard I/O
PyQt5		Clipboard I/O
PyTables	3.4.2	HDF5-based reading / writing
SQLAlchemy	1.1.4	SQL support for databases other than <code>sqlite</code>
SciPy	0.19.0	Miscellaneous statistical functions
XLsxWriter	0.9.8	Excel writing
blosc		Compression for HDF5
fastparquet	0.3.2	Parquet reading / writing

continues on next page

Table 1 – continued from previous page

Dependency	Minimum Version	Notes
gcsfs	0.2.2	Google Cloud Storage access
html5lib		HTML parser for <code>read_html</code> (see note)
lxml	3.8.0	HTML parser for <code>read_html</code> (see note)
matplotlib	2.2.2	Visualization
numba	0.46.0	Alternative execution engine for rolling operations
openpyxl	2.5.7	Reading / writing for xlsx files
pandas-gbq	0.8.0	Google Big Query access
psycopg2		PostgreSQL engine for sqlalchemy
pyarrow	0.12.0	Parquet, ORC (requires 0.13.0), and feather reading / writing
pymysql	0.7.11	MySQL engine for sqlalchemy
pyreadstat		SPSS files (.sav) reading
pytables	3.4.2	HDF5 reading / writing
pyxlsb	1.0.6	Reading for xlsb files
qtpy		Clipboard I/O
s3fs	0.3.0	Amazon S3 access
tabulate	0.8.3	Printing in Markdown-friendly format (see tabulate)
xarray	0.8.2	pandas-like API for N-dimensional data
xclip		Clipboard I/O on linux
xlrd	1.1.0	Excel reading
xlwt	1.2.0	Excel writing
xsel		Clipboard I/O on linux
zlib		Compression for HDF5

Optional dependencies for parsing HTML

One of the following combinations of libraries is needed to use the top-level `read_html()` function:

Changed in version 0.23.0.

- [BeautifulSoup4](#) and [html5lib](#)
- [BeautifulSoup4](#) and [lxml](#)
- [BeautifulSoup4](#) and [html5lib](#) and [lxml](#)
- Only [lxml](#), although see [HTML Table Parsing](#) for reasons as to why you should probably **not** take this approach.

Warning:

- if you install [BeautifulSoup4](#) you must install either [lxml](#) or [html5lib](#) or both. `read_html()` will **not** work with *only* [BeautifulSoup4](#) installed.
- You are highly encouraged to read [HTML Table Parsing gotchas](#). It explains issues surrounding the installation and usage of the above three libraries.

1.4.2 Package overview

pandas is a [Python](#) package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language**. It is already well on its way toward this goal.

pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

The two primary data structures of pandas, *Series* (1-dimensional) and *DataFrame* (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, *DataFrame* provides everything that R’s `data.frame` provides and much more. pandas is built on top of [NumPy](#) and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Here are just a few of the things that pandas does well:

- Easy handling of **missing data** (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be **inserted and deleted** from *DataFrame* and higher dimensional objects
- Automatic and explicit **data alignment**: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let *Series*, *DataFrame*, etc. automatically align the data for you in computations
- Powerful, flexible **group by** functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
- Make it **easy to convert** ragged, differently-indexed data in other Python and NumPy data structures into *DataFrame* objects
- Intelligent label-based **slicing, fancy indexing**, and **subsetting** of large data sets
- Intuitive **merging** and **joining** data sets
- Flexible **reshaping** and pivoting of data sets
- **Hierarchical** labeling of axes (possible to have multiple labels per tick)
- Robust IO tools for loading data from **flat files** (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast **HDF5 format**
- **Time series**-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging.

Many of these principles are here to address the shortcomings frequently experienced using other languages / scientific research environments. For data scientists, working with data is typically divided into multiple stages: munging and cleaning data, analyzing / modeling it, then organizing the results of the analysis into a form suitable for plotting or tabular display. pandas is the ideal tool for all of these tasks.

Some other notes

- pandas is **fast**. Many of the low-level algorithmic bits have been extensively tweaked in [Cython](#) code. However, as with anything else generalization usually sacrifices performance. So if you focus on one feature for your application you may be able to create a faster specialized tool.

- pandas is a dependency of [statsmodels](#), making it an important part of the statistical computing ecosystem in Python.
- pandas has been used extensively in production in financial applications.

Data structures

Dimensions	Name	Description
1	Series	1D labeled homogeneously-typed array
2	DataFrame	General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column

Why more than one data structure?

The best way to think about the pandas data structures is as flexible containers for lower dimensional data. For example, DataFrame is a container for Series, and Series is a container for scalars. We would like to be able to insert and remove objects from these containers in a dictionary-like fashion.

Also, we would like sensible default behaviors for the common API functions which take into account the typical orientation of time series and cross-sectional data sets. When using ndarrays to store 2- and 3-dimensional data, a burden is placed on the user to consider the orientation of the data set when writing functions; axes are considered more or less equivalent (except when C- or Fortran-contiguity matters for performance). In pandas, the axes are intended to lend more semantic meaning to the data; i.e., for a particular data set there is likely to be a “right” way to orient the data. The goal, then, is to reduce the amount of mental effort required to code up data transformations in downstream functions.

For example, with tabular data (DataFrame) it is more semantically helpful to think of the **index** (the rows) and the **columns** rather than axis 0 and axis 1. Iterating through the columns of the DataFrame thus results in more readable code:

```
for col in df.columns:
    series = df[col]
    # do something with series
```

Mutability and copying of data

All pandas data structures are value-mutable (the values they contain can be altered) but not always size-mutable. The length of a Series cannot be changed, but, for example, columns can be inserted into a DataFrame. However, the vast majority of methods produce new objects and leave the input data untouched. In general we like to **favor immutability** where sensible.

Getting support

The first stop for pandas issues and ideas is the [Github Issue Tracker](#). If you have a general question, pandas community experts can answer through [Stack Overflow](#).