Pandas from the Inside

PyData London tutorial, 6 May 2016

Stephen Simmons

mail@stevesimmons.com

https://github.com/SteveSimmons/PyData-PandasFromTheInside

Goals for today…

Foundation

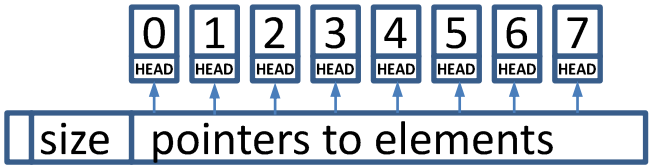
- pandas = Python + numpy + R

Simple analysis (sports statistics), done in depth

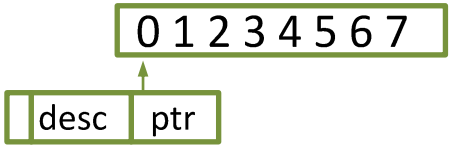
- DataFrames, Slicing, Indexes, GroupBy
- Make sense of syntax
- What is fast vs slow, … and why
- Develop good intuition … for bigger problems

python vs numpy

```
>>> lst = list(range(1000))
[ 0, 1, 2, 3, 4, 5, 6, 7, … ]
```

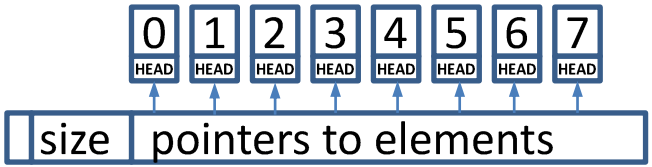



```
>>> import numpy as np

>>> arr = np.arange(1000, dtype=np.int32)
array([ 0, 1, 2, 3, 4, 5, 6, 7, … ], dtype=int32)
```



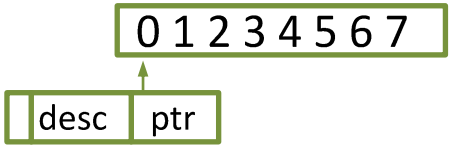
python vs numpy – size

```
>>> import sys
>>> gso = sys.getsizeof

>>> lst = list(range(1000))
[ 0, 1, 2, 3, 4, 5, 6, 7, … ]
```




```
>>> gso([])                          # 64 bytes
>>> gso(0)                           # 24 bytes
>>> gso(lst)                         # 9112 bytes
>>> gso(lst) + sum(gso(x) for x in lst)
                                     # 37108 bytes

>>> hex(id(lst))              # '0x7f1e9c07ed48'
```

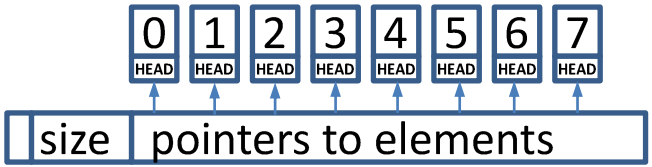
```
>>> import numpy as np

>>> arr = np.arange(1000, dtype=np.int32)
array([ 0, 1, 2, 3, 4, 5, 6, 7, … ], dtype=int32)
```




```
>>> gso(np.array([], dtype=np.int32)) # 96 bytes
>>> arr.itemsize                     # 4
>>> arr.nbytes                       # 4000
>>> gso(arr)                         # 4096 bytes

>>> arr.data          # <memory at 0x7f1e9c14ab88>
>>> arr.shape, arr.strides        # (1000,), (4,)
```

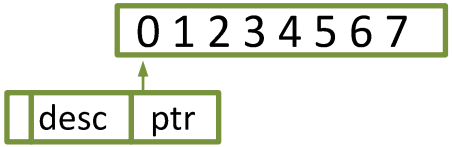
python vs numpy – speed

```
>>> lst = list(range(1000000))
[ 0, 1, 2, 3, 4, 5, 6, 7, … ]
```



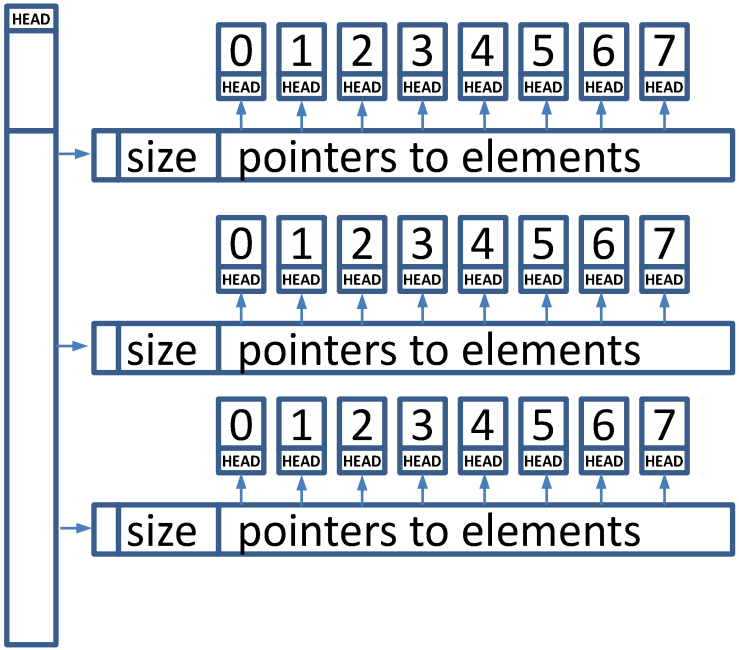

```
>>> %timeit sum(lst)          # 10 ms
```

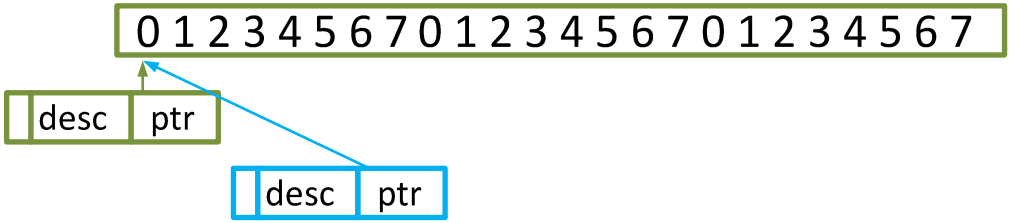
```
>>> arr = np.arange(1000000, dtype='i4')
array([ 0, 1, 2, 3, 4, 5, 6, 7, … ], dtype=int32)
```




```
>>> %timeit arr.sum()         # 1.2 ms
>>> %timeit np.sum(arr)       # 1.2 ms

>>> %timeit sum(arr)          # 150 ms
```

python vs numpy – 2D data

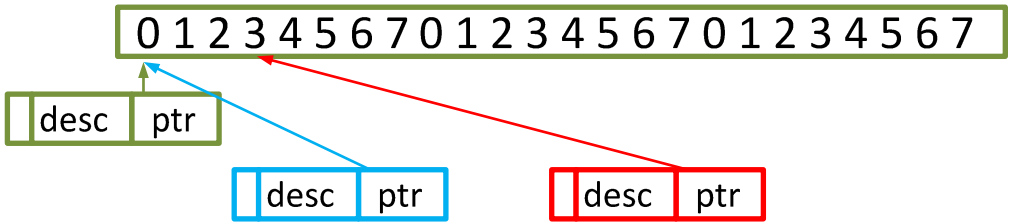





```
>>> arr = np.arange(24, dtype='i4')

>>> arr2 = arr.reshape((3,8))

array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21, 22, 23]], dtype=int32)
```

numpy – slicing and views

```
>>> arr = np.arange(24, dtype='i4')
>>> arr2 = arr.reshape((3,8))
>>> arr3 = arr[3::3]

>>> arr3.base is arr
True
```




```
>>>        np.info(arr), np.info(arr2), np.info(arr3)
class:        ndarray
shape:        (24,)         (3,8)         (7,)
strides:      (4,)          (32,4)        (12,)
itemsize:     4
aligned:      True
contiguous:   True          True          False
fortran:      True          False         False
data pointer: 0x1df3690     0x1df3690     0x1df369c
byteorder:    little
byteswap:     False
type:         int32
```

numpy – indexing

```
>>> arr = np.arange(24, dtype='i4')
>>> arr2 = arr.reshape((3,8))

Scalar index
>>> arr2[1]

Slice
>>> arr3 = arr2[:,0:4]


Integer row indexes (location)
>>> arr4 = arr3.ravel()
>>> arr4[ [1,5,7] ]
>>> arr4.take([1,5,7])

Boolean indexing
>>> arr4 % 3 == 0
>>> arr4[ arr4 % 3 == 0 ]

Assigning to a slice
>>> arr4[ arr4 % 3 == 0 ] = -1
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21, 22, 23]], dtype=int32)

array([ 8,  9, 10, 11, 12, 13, 14, 15], dtype=int32)

array([[ 0,  1,  2,  3],
       [ 8,  9, 10, 11],
       [16, 17, 18, 19]], dtype=int32)


array([0,1,2,3,8,9,10,11,16,17,18,19], dtype=int32)
array([ 1,  9, 11], dtype=int32)
array([ 1,  9, 11], dtype=int32)

array([ True, False, False, True, False, …], dtype=bool)
array([ 0,  3,  9, 18], dtype=int32)

array([-1,1,2,-1,8,-1,10,11,16,17,-1,19], dtype=int32)
```

Pandas = best of Python + numpy + R

Python	- Easy syntax
	- Good for prototyping ("…but slow")
	- Helpful community
Numpy	- Fast, memory-efficient calcs
	- Well-tested algorithms
R	- DataFrame column labels
	- Indexes to align rows

Top-level classes

```
DataFrame
Series
TimeSeries

Index
MultiIndex
Int64Index
Float64Index
RangeIndex

Grouper
Categorical
CategoricalIndex

Timestamp
DatetimeIndex
Timedelta
TimedeltaIndex
Period
PeriodIndex
DateOffset
TimeGrouper

Panel / WidePanel / Panel4D
ExcelFile / ExcelWriter / HDFStore
SparseArray / SparseDataFrame / SparseSeries
Expr / Term
```

Top-level classes

```
DataFrame
Series
TimeSeries

Index
MultiIndex
Int64Index
Float64Index
RangeIndex

Grouper
Categorical
CategoricalIndex

Timestamp
DatetimeIndex
Timedelta
TimedeltaIndex
Period
PeriodIndex
DateOffset
TimeGrouper

Panel / WidePanel / Panel4D
ExcelFile / ExcelWriter / HDFStore
SparseArray / SparseDataFrame / SparseSeries
Expr / Term
```

core.internals

```
SingleBlockManager
BlockManager
BlockPlacement

JoinUnit
NonConsolidatableMixIn

Block
ObjectBlock / BoolBlock
NumericBlock / IntBlock
FloatBlock / ComplexBlock
DatetimeBlock /
TimeDeltaBlock
DatetimeTZBlock /
DatetimeTZDtype
CategoricalBlock
SparseBlock
```

Top-level classes

DataFrame
Series
TimeSeries

Index
MultiIndex
Int64Index
Float64Index
RangeIndex

Grouper
Categorical
CategoricalIndex

Timestamp
DatetimeIndex
Timedelta
TimedeltaIndex
Period
PeriodIndex
DateOffset
TimeGrouper

Panel / WidePanel / Panel4D
ExcelFile / ExcelWriter / HDFStore
SparseDataFrame / SparseSeries
Expr / Term

pd.DataFrame (425 methods!)

T, abs, add, add_prefix, add_suffix, align, all, any, append, apply, applymap, as_blocks, as_matrix, asfreq, assign, astype, at, at_time, axes, between_time, bfill, blocks, bool, boxplot, clip, clip_lower, clip_upper, columns, combine, combineAdd, combineMult, combine_first, compound, consolidate, convert_objects, copy, corr, corrwith, count, cov, cummax, cummin, cumprod, cumsum, describe, diff, div, divide, dot, drop, drop_duplicates, dropna, dtypes, duplicated, empty, eq, equals, eval, ewm, expanding, ffill, fillna, filter, first, first_valid_index, floordiv, from_csv, from_dict, from_items, from_records, ftypes, ge, get, get_dtype_counts, get_ftype_counts, get_value, get_values, groupby, gt, head, hist, iat, icol, idxmax, idxmin, iget_value, iloc, index, info, insert, interpolate, irow, is_copy, isin, isnull, items, iteritems, iterkv, iterrows, itertuples, ix, join, keys, kurt, kurtosis, last, last_valid_index, le, loc, lookup, lt, mad, mask, max, mean, median,, merge, min, mod, mode, mul, multiply, ndim, ne, nlargest, notnull, nsmallest, pct_change, pipe, pivot, pivot_table, plot, pop, pow, prod, product, quantile, query, radd, rank, rdiv, reindex, reindex_axis, reindex_like, rename, rename_axis, reorder_levels, replace, resample, reset_index, rfloordiv, rmod, rmul, rolling, round, rpow, rsub, rtruediv, sample, select, select_dtypes, sem, set_axis, set_index, set_value, shape, shift, size, skew, slice_shift, sort, sort_index, sort_values, sortlevel, squeeze, stack, std, style, sub, subtract, sum, swapaxes, swaplevel, tail, take, to_clipboard, to_csv, to_dense, to_dict, to_excel, to_gbq, to_hdf, to_html, to_json, to_latex, to_msgpack, to_panel, to_period, to_pickle, to_records, to_sparse, to_sql, to_stata, to_string, to_timestamp, to_wide, to_xarray, transpose, truediv, truncate, tshift, tz_convert, tz_localize, unstack, update, values, var, where, xs …

_AXIS_ALIASES, _AXIS_IALIASES, _AXIS_LEN, _AXIS_NAMES, _AXIS_NUMBERS, _AXIS_ORDERS, _AXIS_REVERSED, _AXIS_SLICEMAP, _accessors, _add_numeric_operations, _add_series_only_operations, _add_series_or_dataframe_operations, _agg_by_level, _align_frame, _align_series, _apply_broadcast, _apply_empty_result, _apply_raw, _apply_standard, _at, _box_col_values, _box_item_values, _check_inplace_setting, _check_is_chained_assignment_possible, _check_percentile, _check_setitem_copy, _clear_item_cache, _combine_const, _combine_frame, _combine_match_columns, _combine_match_index, _combine_series, _combine_series_infer, _compare_frame, _compare_frame_evaluate, _consolidate_inplace, _construct_axes_dict, _construct_axes_dict_for_slice, _construct_axes_dict_from, _construct_axes_from_arguments, _constructor, _constructor_expanddim, _constructor_sliced, _convert, _count_level, _create_indexer, _dir_additions, _dir_deletions, _ensure_valid_index, _expand_axes, _flex_compare_frame, _from_arrays, _from_axes, _get_agg_axis, _get_axis, _get_axis_name, _get_axis_number, _get_axis_resolvers, _get_block_manager_axis, _get_bool_data, _get_cacher, _get_index_resolvers, _get_item_cache, _get_numeric_data, _get_values, _getitem_array, _getitem_column, _getitem_frame, _getitem_multilevel, _getitem_slice, _iat, _iget_item_cache, _iloc, _indexed_same, _info_axis, _info_axis_name, _info_axis_number, _info_repr, _init_dict, _init_mgr, _init_ndarray, _internal_names, _internal_names_set, _is_cached, _is_datelike_mixed_type, _is_mixed_type, _is_numeric_mixed_type, _is_view, _ix, _ixs, _join_compat, _loc, _maybe_cache_changed, _maybe_update_cacher, _metadata, _needs_reindex_multi, _nsorted, _protect_consolidate, _reduce, _reindex_axes, _reindex_axis, _reindex_columns, _reindex_index, _reindex_multi, _reindex_with_indexers, _repr_fits_horizontal_, _repr_fits_vertical_, _repr_html_, _repr_latex_, _reset_cache, _reset_cacher, _sanitize_column, _series,

core.internals

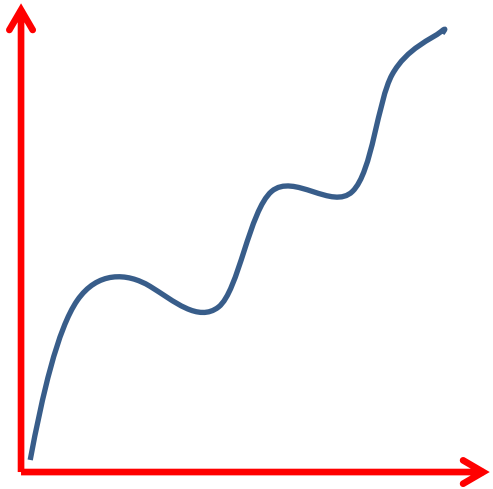
SingleBlockManager
BlockManager
BlockPlacement

JoinUnit
NonConsolidatableMixIn

Block
ObjectBlock / BoolBlock
NumericBlock / IntBlock
FloatBlock / ComplexBlock
DatetimeBlock / TimeDeltaBlock
DatetimeTZBlock / DatetimeTZDtype
CategoricalBlock
SparseBlock

Pandas = a bumpy learning curve

- Broad top-level namespace
- Syntactic sugar
- Large codebase
- Rapid evolution
- API backwards compatible
- Stack Overflow answers
- ‘Big data’



Typical pandas analytical steps

1. Load raw data into DataFrame
2. Reformat columns and add row indexes
3. Select subsets of rows
4. Aggregate and subtotal with GroupBy
5. Post-process for display
6. Compare with other data

Starting point: Aussie Rules football data

All games in chronological order

1.	8-May-1897	R1	Fitzroy	6.13.49	Carlton	2.4.16	Brunswick St
2.	8-May-1897	R1	Collingwood	5.11.41	St Kilda	2.4.16	Victoria Park
3.	8-May-1897	R1	Geelong	3.6.24	Essendon	7.5.47	Corio Oval
…							
14838.	1-May-2016	R6	Brisbane Lions	14.10.94	Sydney	15.7.97	Gabba
14839.	1-May-2016	R6	Carlton	10.12.72	Essendon	8.9.57	M.C.G.
14840.	1-May-2016	R6	West Coast	18.16.124	Collingwood	9.8.62	Subiaco

Goals * 6 + Behinds == Points

http://afltables.com/afl/stats/biglists/bg3.txt

Aussie Rules = footy = football (!= soccer)

End result: premiership ladder

Toyota AFL Premiership Season Ladder

SEASON 2016 ROUND All Rounds Advanced Options

Pos	Club	P	W	L	D	F	A	%	Pts
1	North Melbourne	6	6	0	0	672	538	124.9	24
2	Geelong Cats	6	5	1	0	664	380	174.7	20
3	Sydney Swans	6	5	1	0	645	448	144	20
4	Western Bulldogs	6	4	2	0	536	344	155.8	16
5	GWS Giants	6	4	2	0	670	476	140.8	16
6	West Coast Eagles	6	4	2	0	609	467	130.4	16
7	Adelaide Crows	6	4	2	0	688	564	122	16
8	Hawthorn	6	4	2	0	560	610	91.8	16
9	Melbourne	6	3	3	0	605	592	102.2	12
10	Gold Coast Suns	6	3	3	0	565	595	95	12
11	Port Adelaide	6	3	3	0	549	612	89.7	12
12	St Kilda	6	2	4	0	553	625	88.5	8
13	Collingwood	6	2	4	0	501	637	78.7	8
14	Carlton	6	2	4	0	388	528	73.5	8
15	Richmond	6	1	5	0	482	646	74.6	4
16	Brisbane Lions	6	1	5	0	509	719	70.8	4
17	Essendon	6	1	5	0	363	586	62	4
18	Fremantle	6	0	6	0	430	622	69.1	0

http://www.afl.com.au/ladder

1. Load raw data into DataFrame

All games in chronological order

1.	8-May-1897	R1	Fitzroy	6.13.49	Carlton	2.4.16	Brunswick St
2.	8-May-1897	R1	Collingwood	5.11.41	St Kilda	2.4.16	Victoria Park
3.	8-May-1897	R1	Geelong	3.6.24	Essendon	7.5.47	Corio Oval
…							
14838.	1-May-2016	R6	Brisbane Lions	14.10.94	Sydney	15.7.97	Gabba
14839.	1-May-2016	R6	Carlton	10.12.72	Essendon	8.9.57	M.C.G.
14840.	1-May-2016	R6	West Coast	18.16.124	Collingwood	9.8.62	Subiaco

```
cols = 'GameNum Date Round HomeTeam HomeScore AwayTeam AwayScore Venue'
df = pd.read_csv(filename, skiprows=2, sep='[. ] +',
                 names=cols.split(), parse_dates=['Date'],
                 quoting=csv.QUOTE_NONE, engine='python')


# Or using the sample tutorial code:
>>> import pfi
>>> df = pfi.load_scores()
```

DataFrame structure

```
>>> df
```

	GameNum	Date	Round	HomeTeam	HomeScore	AwayTeam	AwayScore	Venue
0	1	1897-05-08	R1	Fitzroy	6.13.49	Carlton	2.4.16	Brunswick St
1	2	1897-05-08	R1	Collingwood	5.11.41	St Kilda	2.4.16	Victoria Park
2	3	1897-05-08	R1	Geelong	3.6.24	Essendon	7.5.47	Corio Oval
...	...	...	...	...	...	...	...	...
14837	14838	2016-05-01	R6	Brisbane Lions	14.10.94	Sydney	15.7.97	Gabba
14838	14839	2016-05-01	R6	Carlton	10.12.72	Essendon	8.9.57	M.C.G.
14839	14840	2016-05-01	R6	West Coast	18.16.124	Collingwood	9.8.62	Subiaco

```
[14840 rows x 8 columns]

>>> df.index                  # RangeIndex(start=0, stop=14840, step=1)
>>> df.columns                # Index(['GameNum','Date','Round','HomeTeam',…], dtype='object')
>>> df.dtypes.tolist()        # [ dtype('int64'), dtype('<M8[ns]'), dtype('O'), dtype('O'), …]
>>> df.values                 # numpy.array or numpy.recarray

array([[1,     Timestamp('1897-05-08 00:00:00'), 'R1', ..., '2.4.16', 'Brunswick St'],
       [2,     Timestamp('1897-05-08 00:00:00'), 'R1', ..., '2.4.16', 'Victoria Park'],
       [3,     Timestamp('1897-05-08 00:00:00'), 'R1', ..., '7.5.47', 'Corio Oval'],
       ...,
       [14838, Timestamp('2016-05-01 00:00:00'), 'R6', ..., '15.7.97', 'Gabba'],
       [14839, Timestamp('2016-05-01 00:00:00'), 'R6', ...,  '8.9.57', 'M.C.G.'],
       [14840, Timestamp('2016-05-01 00:00:00'), 'R6', ...,  '9.8.62', 'Subiaco']
], dtype=object)
```

Selecting DataFrame columns

```
>>> df
```

	GameNum	Date	Round	HomeTeam	HomeScore	AwayTeam	AwayScore	Venue
0	1	1897-05-08	R1	Fitzroy	6.13.49	Carlton	2.4.16	Brunswick St
1	2	1897-05-08	R1	Collingwood	5.11.41	St Kilda	2.4.16	Victoria Park
2	3	1897-05-08	R1	Geelong	3.6.24	Essendon	7.5.47	Corio Oval
...	...	...	...	...	...	...	...	...
14837	14838	2016-05-01	R6	Brisbane Lions	14.10.94	Sydney	15.7.97	Gabba
14838	14839	2016-05-01	R6	Carlton	10.12.72	Essendon	8.9.57	M.C.G.
14839	14840	2016-05-01	R6	West Coast	18.16.124	Collingwood	9.8.62	Subiaco

```
[14840 rows x 8 columns]
```

```
>>> %timeit df.values                          # 35ms  - numpy.recarray

>>> %timeit df[['Round', 'HomeScore']]         # 500µs – pandas.DataFrame

>>> %timeit df['Round'], df['HomeScore']       # 2µs per col – tuple of pandas.Series

>>> %timeit df['HomeScore']                    # 2µs - pandas.Series
>>> %timeit df[['HomeScore']]                  # 480µs – pandas.DataFrame
```

Behind the scenes: BlockManager

```
>>> df
```

	GameNum	Date	Round	HomeTeam	HomeScore	AwayTeam	AwayScore	Venue
0	1	1897-05-08	R1	Fitzroy	6.13.49	Carlton	2.4.16	Brunswick St
1	2	1897-05-08	R1	Collingwood	5.11.41	St Kilda	2.4.16	Victoria Park
2	3	1897-05-08	R1	Geelong	3.6.24	Essendon	7.5.47	Corio Oval
...	...	...	...	...	...	...	...	...
14837	14838	2016-05-01	R6	Brisbane Lions	14.10.94	Sydney	15.7.97	Gabba
14838	14839	2016-05-01	R6	Carlton	10.12.72	Essendon	8.9.57	M.C.G.
14839	14840	2016-05-01	R6	West Coast	18.16.124	Collingwood	9.8.62	Subiaco

```
[14840 rows x 8 columns]

>>> df.blocks       # Dict with columns with same dtype ->  homogeneous numpy array
{'object': <DF with obj cols>, 'int64': <DF with int64 cols>, 'datetime64[ns]': <DF with TS cols>}

>>> bm = df._data   # BlockManager
>>> bm.blocks
(IntBlock:      slice(0, 1, 1), 1 x 14840, dtype: int64,
 DatetimeBlock: slice(1, 2, 1), 1 x 14840, dtype: datetime64[ns],
 ObjectBlock:   slice(2, 8, 1), 6 x 14840, dtype: object)
>>> bm._blknos, bm._blklocs
(array([0, 1, 2, 2, 2, 2, 2, 2]), array([0, 0, 0, 1, 2, 3, 4, 5]))
>>> df.columns
Index(['GameNum','Date','Round','HomeTeam','HomeScore','AwayTeam','AwayScore','Venue'], dtype='object')

>>> %timeit -n10 df._data.blocks[2].values[2]
array(['6.13.49', '5.11.41', '3.6.24', ..., '14.10.94', '10.12.72', '18.16.124'], dtype=object)
10 loops, best of 3: 338 ns per loop

>>> df['HomeScore'].values.base is df._data.blocks[2].values[2].base
True
```

Plus caching: df[col] → df.__getitem__

```
>>> df.__getitem__??

def __getitem__(self, key):
    # shortcut if we are an actual column
    is_mi_columns = isinstance(self.columns, MultiIndex)
    try:
        if key in self.columns and not is_mi_columns:
            return self._getitem_column(key)
    except:
        pass

    # see if we can slice the rows
    indexer = convert_to_index_sliceable(self, key)
    if indexer is not None:
        return self._getitem_slice(indexer)

    if isinstance(key, (Series, np.ndarray, Index, list)):
        # either boolean or fancy integer index
        return self._getitem_array(key)
    elif isinstance(key, DataFrame):
        return self._getitem_frame(key)
    elif is_mi_columns:
        return self._getitem_multilevel(key)
    else:
        return self._getitem_column(key)

>>> df._getitem_column??

def _getitem_column(self, key):
    """ return the actual column """
    # get column
    if self.columns.is_unique:
        return self._get_item_cache(key)
    # duplicate columns & possible reduce dimensionality
    result = self._constructor(self._data.get(key))
    if result.columns.is_unique:
        result = result[key]
    return result
```

```
>>> df._get_item_cache??

def _get_item_cache(self, item):
    """Return the cached item, item represents a label indexer."""
    cache = self._item_cache
    res = cache.get(item)
    if res is None:
        values = self._data.get(item)
        res = self._box_item_values(item, values)
        cache[item] = res
        res._set_as_cached(item, self)
        # for a chain
        res.is_copy = self.is_copy
    return res
```

```
>>> df._clear_item_cache()

>>> %timeit -n10 df['GameNum']
The slowest run took 6.93 times longer than the fastest.
This could mean that an intermediate result is being cached.
10 loops, best of 3: 1.95µs per loop

>>> df._item_cache
{'GameNum': <series_obj>}

>>> %timeit -n10 df['GameNum']
10 loops, best of 3: 1.88µs per loop
```

Selecting DataFrame columns (2)

```
>>> df
```

	GameNum	Date	Round	HomeTeam	HomeScore	AwayTeam	AwayScore	Venue
0	1	1897-05-08	R1	Fitzroy	6.13.49	Carlton	2.4.16	Brunswick St
1	2	1897-05-08	R1	Collingwood	5.11.41	St Kilda	2.4.16	Victoria Park
2	3	1897-05-08	R1	Geelong	3.6.24	Essendon	7.5.47	Corio Oval
...	...	...	...	...	...	...	...	...
14837	14838	2016-05-01	R6	Brisbane Lions	14.10.94	Sydney	15.7.97	Gabba
14838	14839	2016-05-01	R6	Carlton	10.12.72	Essendon	8.9.57	M.C.G.
14839	14840	2016-05-01	R6	West Coast	18.16.124	Collingwood	9.8.62	Subiaco

```
[14840 rows x 8 columns]
```

```
>>> df[['GameNum', 'Round']]                              # 500us

>>> idx = df.columns.get_indexer(['GameNum','Round'])     # 120us
array([0, 2])

>>> df.take(idx, axis=1)                                  # 360us
```

	GameNum	Round
0	1	R1
1	2	R1
2	3	R1
...	...	...
14837	14838	R6
14838	14839	R6
14839	14840	R6

```
[14840 rows x 2 columns]
```

Terminology: **indexer** = array of integers saying which items to take

Selecting DataFrame rows

```
>>> df
```

	GameNum	Date	Round	HomeTeam	HomeScore	AwayTeam	AwayScore	Venue
0	1	1897-05-08	R1	Fitzroy	6.13.49	Carlton	2.4.16	Brunswick St
1	2	1897-05-08	R1	Collingwood	5.11.41	St Kilda	2.4.16	Victoria Park
2	3	1897-05-08	R1	Geelong	3.6.24	Essendon	7.5.47	Corio Oval
...	...	...	...	...	...	...	...	...
14837	14838	2016-05-01	R6	Brisbane Lions	14.10.94	Sydney	15.7.97	Gabba
14838	14839	2016-05-01	R6	Carlton	10.12.72	Essendon	8.9.57	M.C.G.
14839	14840	2016-05-01	R6	West Coast	18.16.124	Collingwood	9.8.62	Subiaco

```
[14840 rows x 8 columns]

>>> df2 = df[14837:14839]   # See note...

       GameNum       Date Round    HomeTeam HomeScore      AwayTeam AwayScore      Venue
14837    14838 2016-05-01    R6  Brisbane Lions    14.10.94      Sydney   15.7.97      Gabba
14838    14839 2016-05-01    R6         Carlton    10.12.72     Essendon    8.9.57     M.C.G.

>>> %timeit -n10  df[14837:14839]
10 loops, best of 3: 127 µs per loop

>>> df2.columns is df.columns
True
>>> df2.index
RangeIndex(start=14837, stop=14839, step=1)
```

```
Note: compare  df['GameNum'][14837:14839]
          and  df['GameNum'].iloc[14837:14839]
```

Select = index lookup + slice numpy array plus index

```
>>> df
```

	GameNum	Date	Round	HomeTeam	HomeScore	AwayTeam	AwayScore	Venue
0	1	1897-05-08	R1	Fitzroy	6.13.49	Carlton	2.4.16	Brunswick St
1	2	1897-05-08	R1	Collingwood	5.11.41	St Kilda	2.4.16	Victoria Park
2	3	1897-05-08	R1	Geelong	3.6.24	Essendon	7.5.47	Corio Oval
...	...	...	...	...	...	...	...	...
14837	14838	2016-05-01	R6	Brisbane Lions	14.10.94	Sydney	15.7.97	Gabba
14838	14839	2016-05-01	R6	Carlton	10.12.72	Essendon	8.9.57	M.C.G.
14839	14840	2016-05-01	R6	West Coast	18.16.124	Collingwood	9.8.62	Subiaco

```
[14840 rows x 8 columns]


>>> %timeit df[14837:14839]                          # 99 µs per loop

>>> %timeit df['HomeScore'][14837:14839]             # 68 µs per loop  - index lookup

>>> %timeit df['HomeScore'].values[14837:14839]      #  3 µs per loop  - direct slice of numpy array


>>> s = df['HomeScore']

>>> %timeit s[14837:14839]                           # 64 µs per loop

>>> %timeit s.values[14837:14839]                    # 936 ns per loop
```

2. Reformat columns. Add row indexes

```
>>> df
      GameNum       Date Round        HomeTeam  HomeScore     AwayTeam AwayScore          Venue
0           1 1897-05-08    R1         Fitzroy     6.13.49      Carlton     2.4.16   Brunswick St
1           2 1897-05-08    R1     Collingwood     5.11.41     St Kilda     2.4.16  Victoria Park
...       ...        ...   ...             ...         ...          ...        ...            ...
14839   14840 2016-05-01    R6      West Coast  18.16.124  Collingwood     9.8.62        Subiaco
[14840 rows x 8 columns]
```

```
# Using the sample tutorial code:
>>> import pfi
>>> df = pfi.load_scores()
>>> scores = pfi.prepare_scores(df)
```

```
# Convert into sections for both teams
dfi = df.set_index(['Date','Venue','Round'])
home_teams = dfi['HomeTeam'].rename('Team')
away_teams = dfi['AwayTeam'].rename('Team')

# Split scores into Goals/Behinds/pts For and Against
regex = '(?P<G>\d+).(?P<B>\d+).(?P<F>\d+)'
home_scores = dfi['HomeScore']
     .str.extract(regex, expand=True).astype(int)
away_scores = dfi['AwayScore']
     .str.extract(regex, expand=True).astype(int)

home_scores['A'] = away_scores['F']
away_scores['A'] = home_scores['F']

# Combine into new DataFrame
home = pd.concat([home_teams, home_scores], axis=1)
away = pd.concat([away_teams, away_scores], axis=1)

scores = home.append(away)
             .set_index('Team', append=True)
             .sort_index()
```

```
>>> scores
                                                    G   B    F    A
Date       Venue             Round            Team
1897-05-08 Brunswick St      R1             Fitzroy  6  13   49   16
                             R1             Carlton  2   4   16   49
           Corio Oval        R1             Geelong  3   6   24   47
                             R1             Essendon  7   5   47   24
           Lake Oval         R1      South Melbourne  3   9   27   44
                             R1           Melbourne  6   8   44   27
           Victoria Park     R1          Collingwood  5  11   41   16
                             R1             St Kilda  2   4   16   41
1897-05-15 East Melbourne    R2             Essendon  4   6   30   50
...                                                ...  ..  ..  ...  ...
2016-04-30 M.C.G.            R6             Richmond  8  11   59   94
                             R6       Port Adelaide  13  16   94   59
           Sydney Showground R6           GW Sydney  24  14  158   83
                             R6             Hawthorn  12  11   83  158
2016-05-01 Gabba             R6      Brisbane Lions  14  10   94   97
                             R6              Sydney  15   7   97   94
           M.C.G.            R6             Carlton  10  12   72   57
                             R6             Essendon  8   9   57   72
           Subiaco           R6          West Coast  18  16  124   62
                             R6          Collingwood  9   8   62  124
[29680 rows x 5 columns]
```

MultiIndex

```
>>> scores
```

Date	Venue	Round	Team	G	B	F	A
1897-05-08	Brunswick St	R1	Fitzroy	6	13	49	16
		R1	Carlton	2	4	16	49
	Corio Oval	R1	Geelong	3	6	24	47
		R1	Essendon	7	5	47	24
	Lake Oval	R1	South Melbourne	3	9	27	44
		R1	Melbourne	6	8	44	27
	Victoria Park	R1	Collingwood	5	11	41	16
		R1	St Kilda	2	4	16	41
1897-05-15	East Melbourne	R2	Essendon	4	6	30	50
...			...	..	..	...	...
2016-04-30	M.C.G.	R6	Richmond	8	11	59	94
		R6	Port Adelaide	13	16	94	59
	Sydney Showground	R6	GW Sydney	24	14	158	83
		R6	Hawthorn	12	11	83	158
2016-05-01	Gabba	R6	Brisbane Lions	14	10	94	97
		R6	Sydney	15	7	97	94
	M.C.G.	R6	Carlton	10	12	72	57
		R6	Essendon	8	9	57	72
	Subiaco	R6	West Coast	18	16	124	62
		R6	Collingwood	9	8	62	124

```
[29680 rows x 5 columns]
```

```
>>> %timeit df.values
35.3 ms per loop
>>> df._data.blocks
(IntBlock: slice(0, 1, 1), 1 x 14840, dtype: int64,
 DatetimeBlock: slice(1, 2, 1), 1 x 14840, dtype: datetime64[ns],
 ObjectBlock: slice(2, 8, 1), 6 x 14840, dtype: object)
```

```
>>> scores.columns
Index(['G', 'B', 'F', 'A'], dtype='object')

>>> mi = scores.index    # pd.MultiIndex
>>> mi.names
FrozenList(['Date', 'Venue', 'Round', 'Team'])
>>> mi.levels
[ DatetimeIndex(['1897-05-08', …, '2016-05-01'],
    dtype='datetime64[ns]', name='Date', length=4496),
  Index(['Adelaide Oval', 'Albury', 'Arden St',
         'Bellerive Oval', …, 'York Park'],
        dtype='object', name='Venue'),
  Index(['EF', 'GF', 'PF', 'QF', 'R1', 'R10', 'R11',
         …, 'R6', 'R7', 'R8', 'R9', 'SF'],
        dtype='object', name='Round'),
  Index(['Adelaide','Brisbane Bears','Brisbane Lions',
         'Carlton','Collingwood','Essendon',…],
        dtype='object', name='Team')
>>> mi.labels
FrozenList([ [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
1, 1, 2, 2, 2, 2,...], [7, 7, 11, 11, 20, 20, ...],
[4, 4, 4, 4, 4, 4, 4, 4, 15, 15, 15, 15, ...],
[6, 3, 10, 5, 18, 14, 4,...] ])

>>> scores.values
array([[  6,   2,   3, ...,   8,  18,   9],
       [ 13,   4,   6, ...,   9,  16,   8],
       [ 49,  16,  24, ...,  57, 124,  62],
       [ 16,  49,  47, ...,  72,  62, 124]])

>>> %timeit scores.values
5.85 µs per loop

>>> scores._data.blocks
(IntBlock: slice(0, 4, 1), 4 x 29680, dtype: int64,)
```

3. Select subsets of rows

```
>>> scores.loc(axis=0)['2016':'2016', :, 'R1':'R9']
```

Date	Venue	Round	Team	G	B	F	A
2016-03-24	M.C.G.	R1	Richmond	14	8	92	83
			Carlton	12	11	83	92
2016-03-26	Carrara	R1	Gold Coast	17	19	121	60
			Essendon	9	6	60	121
	Docklands	R1	North Melbourne	16	11	107	97
			Adelaide	14	13	97	107
	M.C.G.	R1	Melbourne	12	8	80	78
			GW Sydney	10	18	78	80
	S.C.G.	R1	Sydney	18	25	133	53
			Collingwood	7	11	53	133
...				..	..	...	...
2016-04-30	Sydney Showground	R6	GW Sydney	24	14	158	83
			Hawthorn	12	11	83	158
2016-05-01	Gabba	R6	Brisbane Lions	14	10	94	97
			Sydney	15	7	97	94
	M.C.G.	R6	Carlton	10	12	72	57
			Essendon	8	9	57	72
	Subiaco	R6	West Coast	18	16	124	62
			Collingwood	9	8	62	124

[108 rows x 4 columns]

```
Note: pandas 0.18.0 has a bug if dates are in level 0
      and either first or last dates are not actually in the index:
>>> scores.loc(axis=0)['2015-04-02':'2015-09-06', :, 'R1':'R9']  # 394 rows
>>> scores.loc(axis=0)['2015-04-02':'2015-09-07', :, 'R1':'R9']  # 24577 rows

      Seems to work ok if dates are last level:
>>> scores2 = scores.reorder_levels([1,2,3,0]).sort_index()
>>> scores2.loc(axis=0)[:,'R1':'R9',:,'2015':'2015']             # 394 rows
```

Approach #1 – 820 ms

```
>>> pd.concat( [ sdf  for (dt, rnd), sdf in
            scores.groupby(level=['Date', 'Round'])
          if dt.year == 2016 and rnd.startswith('R')
     ], axis=0)
```

Approach #2 – 1.2 s

```
>>> keep = lambda key: key[0].year==2016
                          and key[2].startswith('R')
>>> scores[ [ keep(key)
              for key, data in scores.iterrows() ] ]
```

Approach #3 – 8 ms

```
>>> scores[ [ keep(key) for key in scores.index ] ]
100 loops, best of 3: 8.29 ms per loop
```

Approach #4 – 12 ms

```
>>> glv = scores.index.get_level_values
>>> scores[ glv('Date').year==2016
            & glv('Round').str.startswith('R') ]
```

Approach #5 – 6 ms

```
>>> scores.loc(axis=0)['2016':'2016', :, 'R1':'R9']
```

DataFrame.loc[] supports in-place updates!

```
>>> scores2 = scores.copy()
>>> scores2.loc(axis=0)['2016':'2016', :, 'R1':'R9'] = 0
>>> scores2
```

Date	Venue	Round	Team	G	B	F	A
1897-05-08	Brunswick St	R1	Fitzroy	6	13	49	16
			Carlton	2	4	16	49
	Corio Oval	R1	Geelong	3	6	24	47
			Essendon	7	5	47	24
	Lake Oval	R1	South Melbourne	3	9	27	44
			Melbourne	6	8	44	27
	Victoria Park	R1	Collingwood	5	11	41	16
			St Kilda	2	4	16	41
1897-05-15	East Melbourne	R2	Essendon	4	6	30	50
			Collingwood	8	2	50	30
...				.	..	..	..
2016-04-30	M.C.G.	R6	Richmond	0	0	0	0
			Port Adelaide	0	0	0	0
	Sydney Showground	R6	GW Sydney	0	0	0	0
			Hawthorn	0	0	0	0
2016-05-01	Gabba	R6	Brisbane Lions	0	0	0	0
			Sydney	0	0	0	0
	M.C.G.	R6	Carlton	0	0	0	0
			Essendon	0	0	0	0
	Subiaco	R6	West Coast	0	0	0	0
			Collingwood	0	0	0	0

[29680 rows x 4 columns]

Behind the scenes…

```
>>> scores2 = scores.copy()
>>> scores2.loc(axis=0)['2016':'2016', :, 'R1':'R9']     # 2.0 ms
>>> scores2.loc(axis=0)['2016':'2016', :, 'R1':'R9'] = 0 # 3.3 ms
>>> scores2
```

				G	B	F	A
Date	Venue	Round	Team				
1897-05-08	Brunswick St	R1	Fitzroy	6	13	49	16
			Carlton	2	4	16	49
	Corio Oval	R1	Geelong	3	6	24	47
			Essendon	7	5	47	24
	Lake Oval	R1	South Melbourne	3	9	27	44
			Melbourne	6	8	44	27
	Victoria Park	R1	Collingwood	5	11	41	16
			St Kilda	2	4	16	41
1897-05-15	East Melbourne	R2	Essendon	4	6	30	50
			Collingwood	8	2	50	30
...				.	..	..	..
2016-04-30	M.C.G.	R6	Richmond	0	0	0	0
			Port Adelaide	0	0	0	0
	Sydney Showground	R6	GW Sydney	0	0	0	0
			Hawthorn	0	0	0	0
2016-05-01	Gabba	R6	Brisbane Lions	0	0	0	0
			Sydney	0	0	0	0
	M.C.G.	R6	Carlton	0	0	0	0
			Essendon	0	0	0	0
	Subiaco	R6	West Coast	0	0	0	0
			Collingwood	0	0	0	0

[29680 rows x 4 columns]

```
>>> tup = ( slice('2016','2016'),
            slice(None),
            slice('R1','R9') )
>>> indexer = scores2.index.get_locs(tup) # 1.34 ms
```

```
array([29572, 29573, 29574, 29575, 29576, 29577, 29578, 29579, 29580,
       29581, 29582, 29583, 29584, 29585, 29586, 29587, 29588, 29589,
       29590, 29591, 29592, 29593, 29594, 29595, 29596, 29597, 29598,
       29599, 29600, 29601, 29602, 29603, 29604, 29605, 29606, 29607,
       29608, 29609, 29610, 29611, 29612, 29613, 29614, 29615, 29616,
       29617, 29618, 29619, 29620, 29621, 29622, 29623, 29624, 29625,
       29626, 29627, 29628, 29629, 29630, 29631, 29632, 29633, 29634,
       29635, 29636, 29637, 29638, 29639, 29640, 29641, 29642, 29643,
       29644, 29645, 29646, 29647, 29648, 29649, 29650, 29651, 29652,
       29653, 29654, 29655, 29656, 29657, 29658, 29659, 29660, 29661,
       29662, 29663, 29664, 29665, 29666, 29667, 29668, 29669, 29670,
       29671, 29672, 29673, 29674, 29675, 29676, 29677, 29678, 29679])
```

```
>>> scores2.ix[indexer]           # 330 µs

>>> scores2.ix[indexer] = 0       # 190 µs
```

4. Add calculated columns

Need to count # of games played, won, drawn, lost

```
>>> y = scores.loc(axis=0)['2016':'2016', :, 'R1':'R9']
>>> y['P'] = 1

SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy

>>> y.is_copy
<weakref at 0x7f3accf35868;
to 'DataFrame' at 0x7f3ace5e3a90>

>>> hex(id(scores))
'0x7f3ace5e3a90'

>>> y = y.copy()

>>> y.is_copy
None

>>> y['P'] = 1      # How fast?
```

Date	Venue	Round	Team	G	B	F	A	P
2016-03-24	M.C.G.	R1	Richmond	14	8	92	83	1
			Carlton	12	11	83	92	1
2016-03-26	Carrara	R1	Gold Coast	17	19	121	60	1
			Essendon	9	6	60	121	1
	Docklands	R1	North Melbourne	16	11	107	97	1
			Adelaide	14	13	97	107	1
	M.C.G.	R1	Melbourne	12	8	80	78	1
...				..	..	...	...	..
2016-04-30	Sydney Showground	R6	Hawthorn	12	11	83	158	1
2016-05-01	Gabba	R6	Brisbane Lions	14	10	94	97	1
			Sydney	15	7	97	94	1
	M.C.G.	R6	Carlton	10	12	72	57	1
			Essendon	8	9	57	72	1
	Subiaco	R6	West Coast	18	16	124	62	1
			Collingwood	9	8	62	124	1

[108 rows x 5 columns]

Inserting a column is fast…

```
>>> %timeit -n1 y['P'] = 1
1 loop, best of 3: 142 µs per loop

>>> y._data.blocks
(IntBlock: slice(0, 4, 1), 4 x 108, dtype: int64,
 IntBlock: slice(4, 5, 1), 1 x 108, dtype: int64)

>>> y._data.is_consolidated()
False

>>> _ = y.max()      # Most funcs trigger consolidation

>>> y._data.is_consolidated()
True

>>> y._data.blocks
(IntBlock: slice(0, 5, 1), 5 x 108, dtype: int64,)
```

Creating the other columns

```
>>> y = y.reset_index(['Date','Venue',
                       'Round'], drop=True)

>>> y['P'] = 1
>>> y['W'] = (y['F'] > y['A'])
>>> y.loc[y['F'] == y['A'], 'D'] = 1
>>> y.eval('L = 1*(A>F)', inplace=True)
```

```
>>> y
```

Team	G	B	F	A	P	W	D	L
Richmond	14	8	92	83	1			
Carlton	12	11	83	92	1			
Gold Coast	17	19	121	60	1			
Essendon	9	6	60	121	1			
North Melbourne	16	11	107	97	1			
Adelaide	14	13	97	107	1			
Melbourne	12	8	80	78	1			
GW Sydney	10	18	78	80	1			
Sydney	18	25	133	53	1			
Collingwood	7	11	53	133	1			
Port Adelaide	20	13	133	100	1			
...	..	..	...	...	..			
Gold Coast	7	6	48	168	1			
Richmond	8	11	59	94	1			
Port Adelaide	13	16	94	59	1			
GW Sydney	24	14	158	83	1			
Hawthorn	12	11	83	158	1			
Brisbane Lions	14	10	94	97	1			
Sydney	15	7	97	94	1			
Carlton	10	12	72	57	1			
Essendon	8	9	57	72	1			
West Coast	18	16	124	62	1			
Collingwood	9	8	62	124	1			

```
[108 rows x 8 columns]
```

Creating the other columns (2)

```
>>> y = y.reset_index(['Date','Venue',
                       'Round'], drop=True)

>>> y['P'] = 1
>>> y['W'] = (y['F'] > y['A'])
>>> y.loc[y['F'] == y['A'], 'D'] = 1
>>> y.eval('L = 1*(A>F)', inplace=True)
```

```
>>> y
```

Team	G	B	F	A	P	W	D	L
Richmond	14	8	92	83	1	True	NaN	0
Carlton	12	11	83	92	1	False	NaN	1
Gold Coast	17	19	121	60	1	True	NaN	0
Essendon	9	6	60	121	1	False	NaN	1
North Melbourne	16	11	107	97	1	True	NaN	0
Adelaide	14	13	97	107	1	False	NaN	1
Melbourne	12	8	80	78	1	True	NaN	0
GW Sydney	10	18	78	80	1	False	NaN	1
Sydney	18	25	133	53	1	True	NaN	0
Collingwood	7	11	53	133	1	False	NaN	1
Port Adelaide	20	13	133	100	1	True	NaN	0
...	..	..	...	...	..	...	..	..
Gold Coast	7	6	48	168	1	False	NaN	1
Richmond	8	11	59	94	1	False	NaN	1
Port Adelaide	13	16	94	59	1	True	NaN	0
GW Sydney	24	14	158	83	1	True	NaN	0
Hawthorn	12	11	83	158	1	False	NaN	1
Brisbane Lions	14	10	94	97	1	False	NaN	1
Sydney	15	7	97	94	1	True	NaN	0
Carlton	10	12	72	57	1	True	NaN	0
Essendon	8	9	57	72	1	False	NaN	1
West Coast	18	16	124	62	1	True	NaN	0
Collingwood	9	8	62	124	1	False	NaN	1

```
[108 rows x 8 columns]
```

Creating the other columns (3)

```
>>> y = y.reset_index(['Date','Venue',
                       'Round'], drop=True)

>>> y['P'] = 1
>>> y['W'] = (y['F'] > y['A']).astype(int)
>>> y.loc[y['F'] == y['A'], 'D'] = 1
>>> y.eval('L = 1*(A>F)', inplace=True)


>>> y['D'] = y['D'].fillna(0)
```

```
>>> y
                 G   B    F    A  P  W    D  L
Team
Richmond        14   8   92   83  1  1  0.0  0
Carlton         12  11   83   92  1  0  0.0  1
Gold Coast      17  19  121   60  1  1  0.0  0
Essendon         9   6   60  121  1  0  0.0  1
North Melbourne 16  11  107   97  1  1  0.0  0
Adelaide        14  13   97  107  1  0  0.0  1
Melbourne       12   8   80   78  1  1  0.0  0
GW Sydney       10  18   78   80  1  0  0.0  1
Sydney          18  25  133   53  1  1  0.0  0
Collingwood      7  11   53  133  1  0  0.0  1
Port Adelaide   20  13  133  100  1  1  0.0  0
...             ..  ..  ...  ... .. .  ... .
Gold Coast       7   6   48  168  1  0  0.0  1
Richmond         8  11   59   94  1  0  0.0  1
Port Adelaide   13  16   94   59  1  1  0.0  0
GW Sydney       24  14  158   83  1  1  0.0  0
Hawthorn        12  11   83  158  1  0  0.0  1
Brisbane Lions  14  10   94   97  1  1  0.0  0
Sydney          15   7   97   94  1  1  0.0  0
Carlton         10  12   72   57  1  1  0.0  0
Essendon         8   9   57   72  1  0  0.0  1
West Coast      18  16  124   62  1  1  0.0  0
Collingwood      9   8   62  124  1  0  0.0  1

[108 rows x 8 columns]
```

Creating the other columns (4)

```
>>> y = y.reset_index(['Date','Venue',
                       'Round'], drop=True)

>>> y['P'] = 1
>>> y['W'] = (y['F'] > y['A']).astype(int)
>>> y.loc[y['F'] == y['A'], 'D'] = 1
>>> y.eval('L = 1*(A>F)', inplace=True)


>>> y['D'] = y['D'].fillna(0)
```

```
>>> y
```

Team	G	B	F	A	P	W	D	L
Richmond	14	8	92	83	1	1	0.0	0
Carlton	12	11	83	92	1	0	0.0	1
Gold Coast	17	19	121	60	1	1	0.0	0
Essendon	9	6	60	121	1	0	0.0	1
North Melbourne	16	11	107	97	1	1	0.0	0
Adelaide	14	13	97	107	1	0	0.0	1
Melbourne	12	8	80	78	1	1	0.0	0
GW Sydney	10	18	78	80	1	0	0.0	1
Sydney	18	25	133	53	1	1	0.0	0
Collingwood	7	11	53	133	1	0	0.0	1
Port Adelaide	20	13	133	100	1	1	0.0	0
...	..	..	...	...	..	..	...	..
Gold Coast	7	6	48	168	1	0	0.0	1
Richmond	8	11	59	94	1	0	0.0	1
Port Adelaide	13	16	94	59	1	1	0.0	0
GW Sydney	24	14	158	83	1	1	0.0	0
Hawthorn	12	11	83	158	1	0	0.0	1
Brisbane Lions	14	10	94	97	1	0	0.0	1
Sydney	15	7	97	94	1	1	0.0	0
Carlton	10	12	72	57	1	1	0.0	0
Essendon	8	9	57	72	1	0	0.0	1
West Coast	18	16	124	62	1	1	0.0	0
Collingwood	9	8	62	124	1	0	0.0	1

```
[108 rows x 8 columns]
```

Creating the other columns (5)

```
>>> y = y.reset_index(['Date','Venue',
                       'Round'], drop=True)

>>> y['P'] = 1
>>> y['W'] = (y['F'] > y['A']).astype(int)
>>> y.loc[y['F'] == y['A'], 'D'] = 1
>>> y.eval('L = 1*(A>F)', inplace=True)


>>> y['D'] = y['D'].fillna(0).astype(int)
Alternatively:
>>> y['D'] = 0
>>> y.loc[y['F'] == y['A'], 'D'] = 1


>>> y._data.blocks
(IntBlock: slice(0, 5, 1), 5 x 108, dtype: int64,
 IntBlock: slice(7, 8, 1), 1 x 108, dtype: int64,
 IntBlock: slice(5, 6, 1), 1 x 108, dtype: int64,
 IntBlock: slice(6, 7, 1), 1 x 108, dtype: int64)
>>> y._data = y._data.consolidate()  # _= y.max()
>>> y._data.blocks
(IntBlock: slice(0, 8, 1), 8 x 108, dtype: int64,)
```

```
>>> y
```

Team	G	B	F	A	P	W	D	L
Richmond	14	8	92	83	1	1	0	0
Carlton	12	11	83	92	1	0	0	1
Gold Coast	17	19	121	60	1	1	0	0
Essendon	9	6	60	121	1	0	0	1
North Melbourne	16	11	107	97	1	1	0	0
Adelaide	14	13	97	107	1	0	0	1
Melbourne	12	8	80	78	1	1	0	0
GW Sydney	10	18	78	80	1	0	0	1
Sydney	18	25	133	53	1	1	0	0
Collingwood	7	11	53	133	1	0	0	1
Port Adelaide	20	13	133	100	1	1	0	0
...	..	..	...	...	..	..	..	..
Gold Coast	7	6	48	168	1	0	0	1
Richmond	8	11	59	94	1	0	0	1
Port Adelaide	13	16	94	59	1	1	0	0
GW Sydney	24	14	158	83	1	1	0	0
Hawthorn	12	11	83	158	1	0	0	1
Brisbane Lions	14	10	94	97	1	0	0	1
Sydney	15	7	97	94	1	1	0	0
Carlton	10	12	72	57	1	1	0	0
Essendon	8	9	57	72	1	0	0	1
West Coast	18	16	124	62	1	1	0	0
Collingwood	9	8	62	124	1	0	0	1

[108 rows x 8 columns]

5. Aggregate/subtotal with GroupBy

```
>>> t = y.groupby(by='Team').sum()

>>> t['PCT'] = 100.0 * t.F / t.A
>>> t['PTS'] = 4 * t['W'] + 2 * t['D']

>>> ladder = t.sort_values(['PTS', 'PCT'],
                           ascending=False)
```

```
>>> ladder
```

Team	G	B	F	A	P	W	D	L	PCT	PTS
North Melbourne	101	66	672	538	6	6	0	0	124.907063	24
Geelong	97	82	664	380	6	5	0	1	174.736842	20
Sydney	94	81	645	448	6	5	0	1	143.973214	20
Western Bulldogs	77	74	536	344	6	4	0	2	155.813953	16
GW Sydney	98	82	670	476	6	4	0	2	140.756303	16
West Coast	88	81	609	467	6	4	0	2	130.406852	16
Adelaide	102	76	688	564	6	4	0	2	121.985816	16
Hawthorn	82	68	560	610	6	4	0	2	91.803279	16
Melbourne	93	47	605	592	6	3	0	3	102.195946	12
Gold Coast	81	79	565	595	6	3	0	3	94.957983	12
Port Adelaide	78	81	549	612	6	3	0	3	89.705882	12
St Kilda	83	55	553	625	6	2	0	4	88.480000	8
Collingwood	74	57	501	637	6	2	0	4	78.649922	8
Carlton	54	64	388	528	6	2	0	4	73.484848	8
Richmond	70	62	482	646	6	1	0	5	74.613003	4
Brisbane Lions	71	83	509	719	6	1	0	5	70.792768	4
Essendon	52	51	363	586	6	1	0	5	61.945392	4
Fremantle	58	82	430	622	6	0	0	6	69.131833	0

Final formatting adjustment

Toyota AFL Premiership Season Ladder

SEASON 2016 ROUND All Rounds Advanced Options

Pos	Club	P	W	L	D	F	A	%	Pts
1	North Melbourne	6	6	0	0	672	538	124.9	24
2	Geelong Cats	6	5	1	0	664	380	174.7	20
3	Sydney Swans	6	5	1	0	645	448	144	20
4	Western Bulldogs	6	4	2	0	536	344	155.8	16
5	GWS Giants	6	4	2	0	670	476	140.8	16
6	West Coast Eagles	6	4	2	0	609	467	130.4	16
7	Adelaide Crows	6	4	2	0	688	564	122	16
8	Hawthorn	6	4	2	0	560	610	91.8	16
9	Melbourne	6	3	3	0	605	592	102.2	12
10	Gold Coast Suns	6	3	3	0	565	595	95	12
11	Port Adelaide	6	3	3	0	549	612	89.7	12
12	St Kilda	6	2	4	0	553	625	88.5	8
13	Collingwood	6	2	4	0	501	637	78.7	8
14	Carlton	6	2	4	0	388	528	73.5	8
15	Richmond	6	1	5	0	482	646	74.6	4
16	Brisbane Lions	6	1	5	0	509	719	70.8	4
17	Essendon	6	1	5	0	363	586	62	4
18	Fremantle	6	0	6	0	430	622	69.1	0

```
>>> pd.options.display.float_format = "%0.1f"
>>> ladder['Pos'] = pd.RangeIndex(1,len(ladder)+1)
>>> ladder
```

Team	G	B	F	A	P	W	D	L	PCT	PTS	Pos
North Melbourne	101	66	672	538	6	6	0	0	124.9	24	1
Geelong	97	82	664	380	6	5	0	1	174.7	20	2
Sydney	94	81	645	448	6	5	0	1	144.0	20	3
Western Bulldogs	77	74	536	344	6	4	0	2	155.8	16	4
GW Sydney	98	82	670	476	6	4	0	2	140.8	16	5
West Coast	88	81	609	467	6	4	0	2	130.4	16	6
Adelaide	102	76	688	564	6	4	0	2	122.0	16	7
Hawthorn	82	68	560	610	6	4	0	2	91.8	16	8
Melbourne	93	47	605	592	6	3	0	3	102.2	12	9
Gold Coast	81	79	565	595	6	3	0	3	95.0	12	10
Port Adelaide	78	81	549	612	6	3	0	3	89.7	12	11
St Kilda	83	55	553	625	6	2	0	4	88.5	8	12
Collingwood	74	57	501	637	6	2	0	4	78.6	8	13
Carlton	54	64	388	528	6	2	0	4	73.5	8	14
Richmond	70	62	482	646	6	1	0	5	74.6	4	15
Brisbane Lions	71	83	509	719	6	1	0	5	70.8	4	16
Essendon	52	51	363	586	6	1	0	5	61.9	4	17
Fremantle	58	82	430	622	6	0	0	6	69.1	0	18

```
# Using the sample tutorial code:
>>> import pfi
>>> df = pfi.load_scores()
>>> scores = pfi.prepare_scores(df)
>>> ladder = pfi.calc_ladder(scores, year=2016)
```

```
def calc_ladder(scores_df, year=2016):
    '''
    DataFrame with championship ladder from round-robin games for the given year.
    Wins, draws and losses are worth 4, 2 and 0 points respectively.
    '''
    # Select a subset of the rows
    # df.loc[] matches dates as strings like '20160506' or '2016'.
    # Note here rounds are simple strings so sort with R1 < R10 < R2 < .. < R9
    #      (we could change this with a CategoricalIndex)
    # Note also that pandas 0.18.0 has a bug with .loc on MultiIndexes
    #      if dates are the first level. It works as expected if we
    #      move the dates to the end before slicing
    scores2 = scores_df.reorder_levels([1, 2, 3, 0]).sort_index()
    x = scores2.loc(axis=0)[:, 'R1':'R9', :, str(year):str(year)]
    # Don't need to put levels back in order as we are about to drop 3 of them
    # x = x.reorder_levels([3, 0, 1, 2]).sort_index()

    # Just keep Team. This does a copy too, avoiding SettingWithCopyWarning
    y = x.reset_index(['Date', 'Venue', 'Round'], drop=True)

    # Add cols with 0/1 for number of games played, won, drawn and lost
    y['P'] = 1
    y['W'] = (y['F'] > y['A']).astype(int)
    y['D'] = 0
    y.loc[y['F'] == y['A'], 'D'] = 1
    y.eval('L = 1*(A>F)', inplace=True)

    # Subtotal by team and then sort by Points/Percentage
    t = y.groupby(level='Team').sum()
    t['PCT'] = 100.0 * t.F / t.A
    t['PTS'] = 4 * t['W'] + 2 * t['D']
    ladder = t.sort_values(['PTS', 'PCT'], ascending=False)

    # Add ladder position (note: assumes no ties!)
    ladder['Pos'] = pd.RangeIndex(1, len(ladder) + 1)

    return ladder
```

GroupBy isn’t necessarily slow

```
>>> %timeit pd.concat( [ subDF.sum()
        for key, subDF in scores.groupby(level=['Venue','Team'])
                        ] )
1 loop, best of 3: 233 ms per loop
```

```
>>> %timeit scores.groupby(level=['Venue','Team']).sum()
100 loops, best of 3: 4.22 ms per loop
```

GroupBy isn’t necessarily slow - internals

```
>>> %timeit scores.groupby(level=['Venue','Team']).sum()
100 loops, best of 3: 4.22 ms per loop


>>> gb = scores.groupby(level=['Venue','Team'])
>>> gb.grouper.groups['M.C.G.', 'Collingwood'][:3]
[(Timestamp('1897-06-19 00:00:00'), 'M.C.G.', 'R6', 'Collingwood'),
…
 (Timestamp('2016-04-25 00:00:00'), 'M.C.G.', 'R5', 'Collingwood')]

>>> idx = gb.grouper.indices['M.C.G.', 'Collingwood']
array([   47,   114,   119,   145,   335,   449,   629,   699,   821,
         826,   919,   968,   985,  1103,  1107,  1199,  1237,  1249,
        1255,  1367,  1391,  1485,  1535,  1707, ...  , 29640, 29660])

>>> %timeit scores['G'][idx].sum()
100 loops, best of 3: 2.78 ms per loop

>>> %timeit scores['G'].values[idx].sum()
The slowest run took 13.73 times longer than the fastest.
This could mean that an intermediate result is being cached.
100000 loops, best of 3: 9.88 µs per loop
```

Conclusions

Pandas is powerful

Lots of ways to get things done
(lots of bad ways too)

Understand its numpy core

See what/how it executes in IPython/Jupyter
(??, %timeit, %prun)

Try scaling up to bigger problems

Don’t be scared to look inside/read the code

Take home challenges …

some ideas for further exploration with pandas

Test your pandas intuition

- Solve a data analysis problem in pandas
- From looking at the code, note where you think time is spent
- Profile it (e.g. in IPython with %prun) to identify where time is actually spent
- Try improvements that are idiomatic pandas, not tricky numpy hacks

HDF5/pytables, bcolz or dask

- Store some pandas DataFrames using several of these systems
- Look at the structure of the stored objects
- Trace what happens when data is loaded into pandas.
- Which are best at minimizing data copying?

Map-reduce

- Take a data analysis problem suitable for solving with map-reduce
- Prototype a direct (non-map) solution using in-memory pandas DataFrames
- Scale up the problem til memory is a concern
- Restructure using map-reduce in pandas
- Can you eliminate many of the expensive index operations?