pandas for Analytics Practitioners, with Applications in Optimization

Dr. Irv Lustig

Optimization Principal

PRINCETON CONSULTANTS

Information Technology and Management Consulting

Question

- You’re working to apply optimization to a new problem

- Do you spend more time
  - Dealing with the data
    - Reading it in, organizing it for optimization, etc.
  - Writing your model
  - Tuning the optimizer’s performance

From Gurobi Webinar on Performance Tuning

- Common support case
  - "Gurobi is soooo slow"
  - Reason: Model building outside of Gurobi takes 30 minutes
  - Model solving takes only a few seconds
- Inefficient data access is the most common reason for slow model construction
  - Long lookup times
  - Insufficient caching / redundant queries
  - Single elements instead of batch processing

http://www.gurobi.com/pdfs/webinars/Webinar-Slides-Introduction-to-Tuning.pdf

What is Python?

- From Wikipedia:
  - **Python** is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.[26]
  - Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.[27]

What is pandas?

- From pandas.pydata.org:
  - *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.
- Library Highlights
  - A fast and efficient **DataFrame** object for data manipulation with integrated indexing;
  - Tools for **reading and writing data** between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format;
  - Intelligent **data alignment** and integrated handling of **missing data**: gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form;
  - Flexible **reshaping** and pivoting of data sets;
  - Intelligent label-based **slicing**, **fancy indexing**, and **subsetting** of large data sets;
  - Columns can be inserted and deleted from data structures for **size mutability**;
  - Aggregating or transforming data with a powerful **group by** engine allowing split-apply-combine operations on data sets;
  - High performance **merging and joining** of data sets;
  - **Hierarchical axis indexing** provides an intuitive way of working with high-dimensional data in a lower-dimensional data structure;
  - **Time series**-functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data;
- Note: Examples in this presentation require pandas v0.22 or later
- Optimization Examples use Princeton Consultants OptiPandas library that extends pandas for use with CPLEX and Gurobi

Outline

- Tidy Data
- pandas Fundamentals
- Reading Data
- Creating New Columns
- Analyzing data
- Reshaping Data
- Merging/joining
- GroupBy
- Optimization

Tidy Data – Defined for Statistics

- "Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types. In tidy data:
  1. Each variable forms a column.
  2. Each observation forms a row.
  3. Each type of observational unit forms a table."
- Hadley Wickham, "Tidy Data", Journal of Statistical Software, Volume 59(10), August 2014. https://www.jstatsoft.org/article/view/v059i10/v59i10.pdf

Tidy and Untidy Data in Optimization (1)

- Untidy data from AMPL (`steelT.*`):

```
set PROD;      # products
param T > 0;   # number of weeks
param revenue {PROD,1..T} >= 0;
param market {PROD,1..T} >= 0;

param T := 4;
set PROD := bands coils;
param revenue:   1     2     3     4 :=
       bands    25    26    27    27
       coils    30    35    37    39 ;
param market:    1     2     3     4 :=
       bands  6000  6000  4000  6500
       coils  4000  2500  3500  4200 ;
```

- Tidy form:

		market	revenue
PROD	**T**		
bands	**1**	6000	25
coils	**1**	4000	30
bands	**2**	6000	26
coils	**2**	2500	35
bands	**3**	4000	27
coils	**3**	3500	37
bands	**4**	6500	27
coils	**4**	4200	39

Tidy and Untidy Data in Optimization (2)

- Untidy data from OPL (knapsack.*)

```
int NbItems = ...;
int NbResources = ...;
range Items = 1..NbItems;
range Resources = 1..NbResources;
int Use[Resources][Items] = ...;

NbResources = 7;
NbItems = 12;
Use = [
      [ 19,   1,  10,  1,   1,  14, 152, 11,  1,   1, 1, 1 ],
      [  0,   4,  53,  0,   0,  80,   0,  4,  5,   0, 0, 0 ],
      [  4, 660,   3,  0,  30,   0,   3,  0,  4,  90, 0, 0],
      [  7,   0,  18,  6, 770, 330,   7,  0,  0,   6, 0, 0],
      [  0,  20,   0,  4,  52,   3,   0,  0,  0,   5, 4, 0],
      [  0,   0,  40, 70,   4,  63,   0,  0, 60,   0, 4, 0],
      [  0,  32,   0,  0,   0,   5,   0,  3,  0, 660, 0, 9]];
```

- Tidy Form:

		Use
Resources	**Items**	
R1	**I1**	19
	I10	1
	I11	1
	I12	1
	I2	1
	I3	10
	I4	1
	I5	1
	I6	14
	I7	152
...	**...**	...
R6	**I3**	40
	I4	70
	I5	4
	I6	63
	I9	60
R7	**I10**	660
	I12	9
	I2	32
	I6	5
	I8	3

Tidy Data – Defined for Optimization

- Wickham Definition for Statistics:

"Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types. In tidy data:

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table."

- Lustig Definition for Optimization:

Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with records, parameters and types. In tidy data:

1. Each parameter forms a column.
2. Each record forms a row.
3. Each type of data unit sharing an index forms a table.
4. A MultiIndex of each record implicitly defines multiple sets

Example of a Table in Tidy Data

		market	revenue
PROD	**T**		
bands	**1**	6000	25
coils	**1**	4000	30
bands	**2**	6000	26
coils	**2**	2500	35
bands	**3**	4000	27
coils	**3**	3500	37
bands	**4**	6500	27
coils	**4**	4200	39

Parameters

1. **Each parameter forms a column.**
2. **Each record forms a row.**
3. **Each type of data unit sharing an index forms a table.**
4. **A MultiIndex of each record implicitly defines multiple sets**

MultiIndex

```
('bands', 1)
('coils', 1)
('bands', 2)
('coils', 2)
('bands', 3)
('coils', 3)
('bands', 4)
('coils', 4)
```

```
PROD = ['bands', 'coils']

T = [1, 2, 3, 4]
```

Tidy Data Read Into Pandas

- tidysteel.csv

```
PROD,T,market,revenue
bands,1,6000,25
coils,1,4000,30
bands,2,6000,26
coils,2,2500,35
bands,3,4000,27
coils,3,3500,37
bands,4,6500,27
coils,4,4200,39
```

01.tidysteel.ipynb

pandas Fundamentals

pandas Series

- Essentially a vector of values (or objects)
- Indexed sequentially (default) and possibly by a specified set of labels

- Examples
  - Create a simple series, which will get indexed sequentially

    ```
    s1 = pd.Series([3,1,4,1,5,9,2,6])
    ```

    - Get a specific element: `s1[5]`

  - Create a series indexed by letters in the alphabet.

    ```
    s2 = pd.Series([0.1*i for i in range(10)],
                   index=list(string.ascii_uppercase)[:10])
    ```

    - Get a specific element: `s2['F']`

  - Create a series of random numbers, indexed by a MultiIndex.

    ```
    s3 = pd.Series(np.random.randn(6),
                   index=pd.MultiIndex
                         .from_product([['A','B'],
                                        [4,5,6]]))
    ```

    - Get a specific element: `s3['B',5]`

Jupyter

02.Series.ipynb

Indexing for Series

- A simple index can be built from any list
  - This is called indexing by *label*
  - Duplicate labels are allowed, but be careful!
- You can pick slices of the index
- Examples
  - Get a mid-sequence of elements
    `s1[2:4]`
  - Get elements from one point to the end
    `s2['G':]`
  - Get elements up to a specific point
    `s2[:'C']`
  - Use integer index to get values from label indexed Series
    `s2[5:8]`
  - Get specified subset of rows
    `s2[['B','E','G']]`

Avoiding Ambiguity with integer Index

- If the index is integer, and not sequential, then referring to specific elements can be ambiguous
  - Use `.loc` or `.iloc` instead
    - `.loc` uses the labels for indexing
    - `.iloc` uses the positional index (row number)
- Example

```
s4 = pd.Series([10*i for i in range(10)],
               index=[10-i for i in range(10)])
```

  - What is `s4[8]` ???
    - pandas will return 20

`s4.loc[8]` is 20

`s4.iloc[8]` is 80

10	0
9	10
8	20
7	30
6	40
5	50
4	60
3	70
2	80
1	90

MultiIndex

- An index with dimension, consisting of *levels*
- Think of it as a set of tuples
- Three equivalent ways of creating a MultiIndex

```
mi1 = pd.MultiIndex
        .from_product([['a','b','c'], [1,2]])
mi2 = pd.MultiIndex
        .from_tuples([('a', 1), ('a', 2), ('b', 1),
                      ('b', 2), ('c', 1), ('c', 2)])
mi3 = pd.MultiIndex
        .from_arrays([['a','a','b','b','c','c'],
                      [1,2,1,2,1,2]])
```

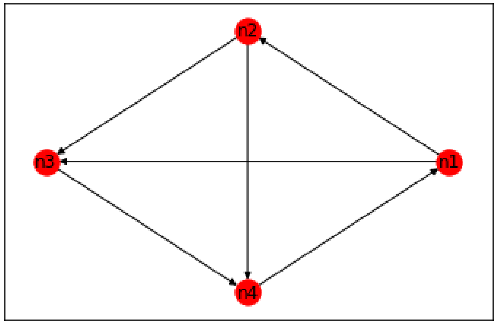
03.MultiIndex.ipynb

Slices with MultiIndex

<table>
<tr><th>Code</th><th>Result</th><th>Description</th></tr>
<tr><td><pre>s1 = pd.Series(
        range(6),
        index=mi1)</pre></td><td><table><tr><td rowspan="2">a</td><td>1</td><td>0</td></tr><tr><td>2</td><td>1</td></tr><tr><td rowspan="2">b</td><td>1</td><td>2</td></tr><tr><td>2</td><td>3</td></tr><tr><td rowspan="2">c</td><td>1</td><td>4</td></tr><tr><td>2</td><td>5</td></tr></table></td><td>Create the series, using the MultiIndex as the index</td></tr>
<tr><td><pre>s1['a']</pre></td><td><table><tr><td>1</td><td>0</td></tr><tr><td>2</td><td>1</td></tr></table></td><td>Get all rows where first label is 'a'</td></tr>
<tr><td><pre>s1[:,2]</pre></td><td><table><tr><td>a</td><td>1</td></tr><tr><td>b</td><td>3</td></tr><tr><td>c</td><td>5</td></tr></table></td><td>Get all rows where second label is 2</td></tr>
</table>

Network Application

```
mi4 = pd.MultiIndex.from_tuples(
          [('n1','n2'),('n1','n3'),
           ('n2','n4'),('n3','n4'),
           ('n2','n3'),('n4','n1')],
          names=['from','to'])
```



Code	Result	Description
`s4 = pd.Series( range(6), index=mi4)`	n1 n2 0 n3 1 n2 n4 2 n3 n4 3 n2 n3 4 n4 n1 5	Create a series indexed by the arc (from,to) pairs
`s4['n1']`	n2 0 n3 1	Get all arcs from n1
`s4['n2']`	n4 2 n3 4	Get all arcs from n2
`s4[:,'n3']`	n1 1 n2 4	Get all arcs into n3

DataFrame

- A pandas DataFrame is a table. Each column is a Series, and each Series is sharing the same Index/MultiIndex

Specify Each Column (Series)	Specify Each Row (Record)

`df1 = pd.DataFrame(`
`{'cost':  [10*i for i in range(6)],`
` 'bound': [100*i for i in range(6)]`
`},       index=mi4)`
	`df2 = pd.DataFrame.from_records(`
	`  [(10*i, 100*i) for i in range(6)],`
	`       columns=['cost','bound'],`
	`       index=mi4)`

		bound	cost
from	**to**		
n1	**n2**	0	0
	n3	100	10
n2	**n4**	200	20
n3	**n4**	300	30
n2	**n3**	400	40
n4	**n1**	500	50

		cost	bound
from	**to**		
n1	**n2**	0	0
	n3	10	100
n2	**n4**	20	200
n3	**n4**	30	300
n2	**n3**	40	400
n4	**n1**	50	500

Q: Anything interesting about these 2 results?

Indexing for DataFrames

- You can get an individual column in one of two ways

```
df1.cost
df1['cost']
```

- Because of the second method, getting a subset of the rows is trickier

04.DataFrame.ipynb

Indexing Options and Errors

Doesn't work	Error reported	Works
`df1['n1']`	`KeyError: 'n1'`	`df1.loc['n1']`
`df1['n1',:]`	`TypeError: unhashable type: 'slice'`	`df1.loc['n1',:]`
`df1.loc[:,'n3']`	`KeyError: 'the label [n3] is not in the [columns]'`	`df1.loc[('n1','n3'), 'cost']`

- Solution for general slicing

```
idx=pd.IndexSlice
```

`df1.loc[idx[:,'n3'], 'cost']`	`KeyError: 'MultiIndex Slicing requires the index to be fully lexsorted tuple len (2), lexsort depth (0)'`	`df1.loc[idx['n1',:]]`

- Sort the index

```
dfs = df1.sort_index() # Make new DataFrame
df1.sort_index(inplace=True) # Same DataFrame
```

`dfs.loc[idx[:,'n3']]`	`KeyError: 'the label [n3] is not in the [columns]'`	`dfs.loc[idx[:,'n3'], 'cost']`

DataFrame Indexing Recommendations

- Get one column

```
df1.cost
```

- Get more than one column

```
df1[['cost','bound']]
```

- To get a slice of rows, use `IndexSlice` and sort the index.

```
idx = pd.IndexSlice
df1.sort_index(inplace=True)
```

- Use `.loc` and specify the columns

```
df1.loc[idx[:,'n3'],'cost']
df1.loc[idx[:,'n3'],:]
```

- Or pick a column (it becomes a Series) and avoid `.loc`

```
df1.cost['n1',:]
df1.cost[:,'n3']
```

Reading Data into DataFrames

CSV and Excel Files

- pandas `read_csv()` and `read_excel()` functions have lots of options
- Default for CSV: `df = pd.read_csv('myfile.csv')`
  - Assume a comma separator
  - Assume there is a header row defining column names
  - Creates a sequential index for the DataFrame
  - Uses all columns and all rows
- Default for Excel: `df = pd.read_excel('myfile.xlsx')`
  - Read the first sheet in the workbook
  - Assume the first row contains names of columns
  - Creates a sequential index for the DataFrame
  - Uses all columns and rows that have values
    - Be careful with a sheet that has extraneous data!

- Various parameters allow you to override the default behaviors

Read CSV Data Into Pandas

- tidysteel.csv

```
PROD,T,market,revenue
bands,1,6000,25
coils,1,4000,30
bands,2,6000,26
coils,2,2500,35
bands,3,4000,27
coils,3,3500,37
bands,4,6500,27
coils,4,4200,39
```

01.tidysteel.ipynb

```
mrdf = pd.read_csv('01.tidysteel.csv',
                   index_col=[0,1])
```

		market	revenue
PROD	**T**		
bands	**1**	6000	25
coils	**1**	4000	30
bands	**2**	6000	26
coils	**2**	2500	35
bands	**3**	4000	27
coils	**3**	3500	37
bands	**4**	6500	27
coils	**4**	4200	39

Read Database Data into pandas

- Consider a MySQL table called "gas" in a schema called "pandas"

name	demand	price	octane	lead
Super	3000.0	70.0	10.0	1.0
Regular	2000.0	60.0	8.0	2.0
Diesel	1000.0	50.0	6.0	1.0

- Use python package `sqlalchemy` to get connection to database (database specific)

```
import sqlalchemy as sqla
engine = sqla.create_engine(
    'mysql+pymysql://irv:password@localhost:3306')
conn=engine.connect()
```

Multiple ways to read the table

- Read all columns, no index created

```
df1 = pd.read_sql_table("gas", con=conn, schema="pandas")
```

- Read all columns, specify column for index

```
df2 = pd.read_sql_table("gas", con=conn, schema="pandas",
                        index_col="name")
```

05.Database.ipynb

- Look up metadata to get primary key information

```
pkeys = (sqla.Table('gas', sqla.MetaData(), autoload=True,
                    autoload_with=engine, schema='pandas')
         .primary_key.columns.keys()) # Get the primary keys
df3 = pd.read_sql_table("gas", con=conn, schema="pandas",
                        index_col=pkeys)
```

- You can also do classic SQL queries

```
df4 = pd.read_sql("SELECT name, demand, price from pandas.gas",
                  conn, index_col="name")
```

Read HTML table data

- Pandas can read data from HTML tables
- May need to figure out where the table is within the HTML page.
- Using example from Gurobi:

  http://www.gurobi.com/resources/examples/food-manufacture-II

```
df = pd.read_html("http://www.gurobi.com/resources/examples/food-manufacture-II")[1]
```

- Note that a list is returned. Have to get the right table.

06.html.ipynb

Problem Description

A manufacturer needs to refine several raw oils and blend them together to produce a given food product that can be sold. The raw oils needed can be divided into two categories:

Category	Oil
Vegetable oils:	VEG 1
	VEG 2
Non-vegetable oils:	OIL 1
	OIL2
	OIL 3

The manufacturer can choose to buy raw oils for the current month and/or buy them on the futures market for delivery in a subsequent month. Prices for immediate deliver and in the futures market are given below in $/ton:

Month	VEG 1	VEG 2	OIL 1	OIL 2	OIL 3
January	110	120	130	110	115
February	130	130	110	90	115
March	110	140	130	100	95
April	120	110	120	120	125
May	100	120	150	110	105
June	90	100	140	80	135

More on I/O supported by pandas

Reading Functions	Writing Functions
`read_csv`	`to_csv`
`read_excel`	`to_excel`
`read_hdf`	`to_hdf`
`read_sql`	`to_sql`
`read_json`	`to_json`
`read_html`	`to_html`
`read_stata`	`to_stata`
`read_sas`	
`read_clipboard`	`to_clipboard`
`read_pickle`	`to_pickle`

Manipulating Data

Creating Indexes

- All the reading functions allow you to specify which columns make up the Index/MultiIndex
- But you can also set the index if there is not one

```
df1 = df.set_index(['Month'])
```

- You can also reset the index to be numerical

```
df2 = df1.reset_index()
```

- If you want to change the index from one column to another, reset and then set:

```
df3 = df1.reset_index().set_index(['VEG 1'])
```

06.html.ipynb

Read the Data, then get the index sets!

- Get the complete index set as a list

```
list(mrdf.index)
```

- Look at the unique values corresponding to the index name 'PROD’ (two methods)

```
set(mrdf.index.get_level_values('PROD’))
mrdf.index.levels[mrdf.index.names.index('PROD')]
```

- Look at the unique values corresponding to the index name ‘T’ (two methods)

```
set(mrdf.index.get_level_values('T’))
```

01.tidysteel.ipynb

Creating New Columns

- Given a DataFrame, you can create new columns that are arithmetic combinations of other columns
  - The computations are done treating Series as vectors
- Example:

```
df = df.assign(totmin = 60*df.hours + df.minutes)
```

- Alternative:

```
df['totmin'] = 60*df.hours + df.minutes
```

	hours	minutes
0	5	14
1	2	4
2	5	7
3	9	51
4	9	1
5	5	14
6	5	51
7	3	1

	hours	minutes	totmin
0	5	14	314
1	2	4	124
2	5	7	307
3	9	51	591
4	9	1	541
5	5	14	314
6	5	51	351
7	3	1	181

07.assign.ipynb

Example Using Date Data

- 33850 records of Begin/End Dates "dates.csv"

```
Begin,End
20150407,20150411
20150404,20150411
20150409,20150413
20150409,20150417
20150409,20150411
20150403,20150411
20150403,20150409
20150403,20150408
20150403,20150403
```

Multiple Transformations with .assign

```
ddf = didf.assign(delta = didf.End-didf.Begin,
                  BegYear = didf.Begin//10000,
                  EndYear = didf.End//10000)
```

Jupyter Nb 07.assign.ipynb

```
ddf = (
    ddf.assign(BegMonth = (ddf.Begin-ddf.BegYear*10000)//100,
               EndMonth = (ddf.End-ddf.EndYear*10000)//100,
               BegDay = ddf.Begin % 100,
               EndDay = ddf.End % 100)
    [['Begin','BegYear','BegMonth','BegDay','End',
      'EndYear','EndMonth','EndDay','delta']]
      )
```

- Note use of multiple assignments, and chaining to reorder the columns

	Begin	BegYear	BegMonth	BegDay	End	EndYear	EndMonth	EndDay	delta
0	20150407	2015	4	7	20150411	2015	4	11	4
1	20150404	2015	4	4	20150411	2015	4	11	7

pandas has built-in Date/Time handling

```
dtdf = didf.assign(
    BegTime = pd.to_datetime(didf.Begin, format='%Y%m%d'),
    EndTime = pd.to_datetime(didf.End, format='%Y%m%d')
    )
```

07.assign.ipynb

```
dtdf = (dtdf.assign(BegYear = dtdf.BegTime.dt.year,
                    BegMonth = dtdf.BegTime.dt.month,
                    BegDay = dtdf.BegTime.dt.day,
                    EndYear = dtdf.EndTime.dt.year,
                    EndMonth = dtdf.EndTime.dt.month,
                    EndDay = dtdf.EndTime.dt.day,
                    delta = dtdf.EndTime - dtdf.BegTime)
        [['Begin','BegTime','BegYear','BegMonth','BegDay',
          'End','EndTime','EndYear','EndMonth','EndDay','delta']]
       )
```

	Begin	BegTime	BegYear	BegMonth	BegDay	End	EndTime	EndYear	EndMonth	EndDay	delta
0	20150407	2015-04-07	2015	4	7	20150411	2015-04-11	2015	4	11	4 days
1	20150404	2015-04-04	2015	4	4	20150411	2015-04-11	2015	4	11	7 days

```
dtdf.dtypes
```

Begin	int64	**End**	int64
BegTime	datetime64[ns]	**EndTime**	datetime64[ns]
BegYear	int64	**EndYear**	int64
BegMonth	int64	**EndMonth**	int64
BegDay	int64	**EndDay**	int64
		delta	timedelta64[ns]

Understanding Data

- After you read in a table, there are useful features to look at parts of the data

Method	Result
`df.head(n)`	First n (default 5) rows of DataFrame
`df.tail(n)`	Last n (default 5) rows of DataFrame
`df.describe()`	Descriptive statistics of each column
`len(df)`	Number of rows in DataFrame
`df.sum()`	Sum of each column
`df.min()`	Minimum value of each column
`df.max()`	Maximum value of each column
`df.hist(bins=n)`	Histogram plot of each column with n bins (default 10)

You can filter and sort data

- Filtering

```
df[df.normal < -25]
df[(df.uniform - df.normal > 7) & (df.normal > 0)]
```

- Sorting

```
df.sort_values(by='normal', inplace=True)
```

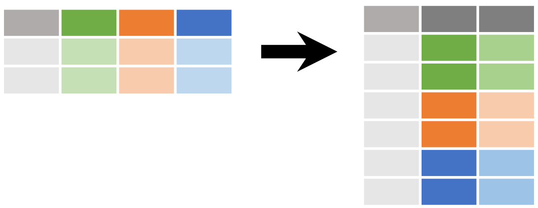
- Note how the index is preserved when sorting!

08.Analyze.ipynb

Other Data Transformations

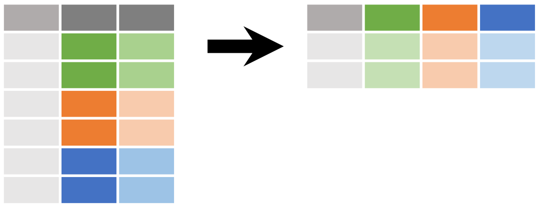
- Melting Helps Make Tidy Data!

```
pd.melt(df, id_vars='rowindex',
        value_name='valname')
```



- Pivoting Does the Reverse

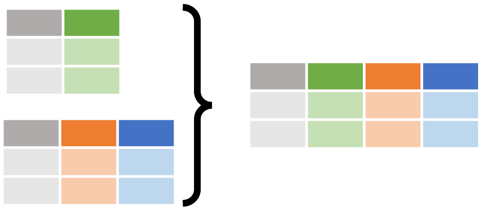
```
df.pivot(index='index',
         columns='var',
         values='val')
```



- You can concatenate rows or columns

```
pd.concat([df1,df2])
```

```
pd.concat([df1,df2], axis=1)
```

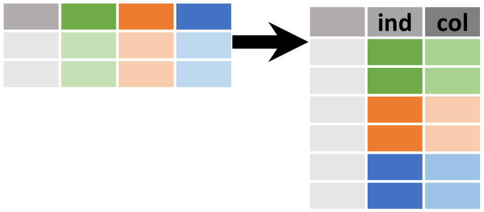


09.Transform.ipynb

Stack/Unstack – Another pivot operation

- Stacking, with renaming, helps make tidy data by transforming 2 dimensional data by moving column names to index (convert wide to long)

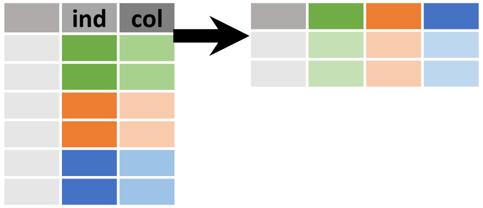
```
df.rename_axis("ind", axis="columns")
   .stack()
   .to_frame("col")
```



- Unstacking is the reverse operation – Good for two dimensional output (convert long to wide)

```
df.unstack()
```

  – Note that missing values become NaN



Merging/Joining

- pandas Supports Database-style Joins and Merges
  - Essential for combining different tables across indices

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

```
pd.merge(adf, bdf,
         how='left', on='x1')
```

Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

```
pd.merge(adf, bdf,
         how='right', on='x1')
```

Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

```
pd.merge(adf, bdf,
         how='inner', on='x1')
```

Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

```
pd.merge(adf, bdf,
         how='outer', on='x1')
```

Join data. Retain all values, all rows.

Merging/Joining Example

personhoursperday		hours
id	day	
ADK0000000000ICFPT	Mo	0
BEL0000000001JDGQU	Mo	8
CFM0000000002KEHRV	Mo	0
DGN0000000003LFISW	Mo	0
EHO0000000004MGJTX	Mo	0

prefs table		pref
person_id	site_id	
MPW0000000272UORBF	ADK0000000364ICFPT	64
ORY0000000196WQTDH	ADK0000000364ICFPT	32
QTA0000000146YSVFJ	ADK0000000364ICFPT	97
TWD0000000123BVYIM	ADK0000000364ICFPT	57
ADK0000000000ICFPT	ADK0000000390ICFPT	35

sitehoursperday		hours
id	day	
ADK0000000364ICFPT	Mo	8.0
	Tu	8.0
	We	8.0
	Th	8.0
	Fr	4.0

	person_id	site_id	pref	day	daily_hours
0	MPW0000000272UORBF	ADK0000000364ICFPT	64	Mo	8.0
1	MPW0000000272UORBF	ADK0000000364ICFPT	64	Tu	8.0
2	MPW0000000272UORBF	ADK0000000364ICFPT	64	We	8.0
3	MPW0000000272UORBF	ADK0000000364ICFPT	64	Th	4.0
4	MPW0000000272UORBF	ADK0000000364ICFPT	64	Fr	4.0
9	ORY0000000196WQTDH	ADK0000000364ICFPT	32	We	4.0
10	ORY0000000196WQTDH	ADK0000000364ICFPT	32	Th	4.0
11	ORY0000000196WQTDH	ADK0000000364ICFPT	32	Fr	4.0
14	QTA0000000146YSVFJ	ADK0000000364ICFPT	97	Mo	6.0
15	QTA0000000146YSVFJ	ADK0000000364ICFPT	97	Tu	6.0
16	QTA0000000146YSVFJ	ADK0000000364ICFPT	97	We	6.0
17	QTA0000000146YSVFJ	ADK0000000364ICFPT	97	Th	6.0
21	TWD0000000123BVYIM	ADK0000000364ICFPT	57	Mo	6.0
22	TWD0000000123BVYIM	ADK0000000364ICFPT	57	Tu	6.0

pandas Merging/Joining Example

peoplehoursperday		hours
id	day	
ADK0000000000ICFPT	Mo	0
BEL0000000001JDGQU	Mo	8
CFM0000000002KEHRV	Mo	0
DGN0000000003LFISW	Mo	0
EHO0000000004MGJTX	Mo	0

prefs table		pref
person_id	site_id	
MPW0000000272UORBF	ADK0000000364ICFPT	64
ORY0000000196WQTDH	ADK0000000364ICFPT	32
QTA0000000146YSVFJ	ADK0000000364ICFPT	97
TWD0000000123BVYIM	ADK0000000364ICFPT	57
ADK0000000000ICFPT	ADK0000000390ICFPT	35

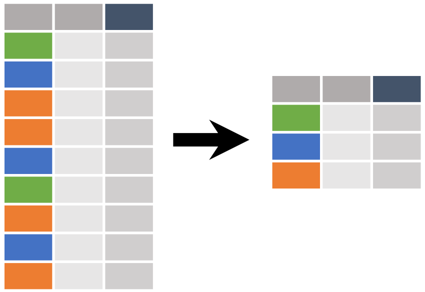
sitehoursperday		hours
id	day	
ADK0000000364ICFPT	Mo	8.0
	Tu	8.0
	We	8.0
	Th	8.0
	Fr	4.0

```
prefswithdaily = (
    pd.merge(prefs.reset_index(),
             peoplehoursperday.reset_index(),
             on='person_id', how='left')
    .rename(columns={'hours' : 'person_hours'})
    .merge(sitehoursperday.reset_index(),
           on=['site_id','day'], how='inner')
    .rename(columns={'hours' : 'site_hours'})
    )
```

10.Merging.ipynb

groupby – A Powerful pandas Operator

- When working with data, we often need to apply a mathematical operator (sum, min, max, etc.) across a column, grouped by other sets of columns



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column or index level named "col".

Function	Description	Function	Description
`size()`	Size of each group	`sum()`	Sum values of each group
`min()`	Minimum of each group	`max()`	Maximum of each group
`mean()`	Mean of each group	`agg(function)`	Apply function to each group

groupby Examples

peoplehoursperday		hours
id	**day**	
ADK0000000000ICFPT	**Mo**	0
BEL0000000001JDGQU	**Mo**	8
CFM0000000002KEHRV	**Mo**	0
DGN0000000003LFISW	**Mo**	0
EHO0000000004MGJTX	**Mo**	0

```
peoplehoursperday
   .groupby(by='id').sum()
```

	hours
id	
ADK0000000000ICFPT	6
ADK0000000026ICFPT	8
ADK0000000052ICFPT	16
ADK0000000078ICFPT	16
ADK0000000104ICFPT	25

```
peoplehoursperday
   .groupby(by='day').sum()
```

	hours
day	
Fr	1110
Mo	1518
Sa	133
Su	88
Th	1696
Tu	1818
We	1757

11.GroupBy.ipynb

pandas for Optimization

OptiPandas

- Princeton Consultants Library for extending pandas for use in optimization modeling
- Allows constraints to be represented as
  - Series(Linear Expression) <= Series(Linear Expression)
  - Series(Linear Expression == Series(Linear Expression)
  - Series(Linear Expression) >= Series(Linear Expression)
- Makes it easier to write sums over MultiIndex expressions in a highly performant way

```
import optipandas as opd
```

Comparing AMPL to pandas

- Model File

```
set PROD;
param T > 0;

param rate {PROD} > 0;
param avail {1..T} >= 0;
```

- Data File

```
param T := 4;
set PROD := bands coils;

param avail :=  1 40  2 40  3 32  4 40 ;

param rate :=  bands 200   coils 140 ;
```

Jupyter Nb

12.datacompare.ipynb

- avail.csv

```
Period,avail
1,40
2,40
3,32
4,40
```

- rate.csv

```
PROD,rate
bands,200
coils,140
```

- Code

```
avail = pd.read_csv('avail.csv',
                    index_col=0)
rate = pd.read_csv('rate.csv',
                   index_col=0)
T = max(avail.index.get_level_values(0))
PROD = set(rate.index.get_level_values(0))
```

- Note how the sets are inferred from the data

AMPL and pandas Data Representation

- AMPL ".dat" file

```
param T := 4;
set PROD := bands coils;

param avail :=  1 40  2 40  3 32  4 40 ;

param rate :=  bands 200   coils 140 ;
param inv0 :=  bands  10   coils   0 ;

param prodcost :=  bands 10    coils  11 ;
param invcost  :=  bands  2.5  coils   3 ;

param revenue:   1     2     3     4 :=
       bands    25    26    27    27
       coils    30    35    37    39 ;

param market:    1     2     3     4 :=
       bands  6000  6000  4000  6500
       coils  4000  2500  3500  4200 ;
```

- Excel for pandas

	A	B	C	D	E
1	***Product***	***rate***	***inv0***	***prodcost***	***invcost***
2	bands	220	10	10	2.5
3	coils	154	0	11	3
4					
5					
6					
7	***Revenue***	***1***	***2***	***3***	***4***
8	***bands***	25	26	27	27
9	***coils***	30	35	37	39
10					
11	***Market***	***1***	***2***	***3***	***4***
12	***bands***	6000	6000	4000	6500
13	***coils***	4000	2500	3500	4200
14					
15					
16					
17	***Time***	***avail***			
18	***1***	40			
19	***2***	40			
20	***3***	32			
21	***4***	40			

AMPL and pandas data declarations

- AMPL declarations

```
set PROD;
param T > 0;

param rate {PROD} > 0;
param inv0 {PROD} >= 0;
param avail {1..T} >= 0;
param market {PROD,1..T} >= 0;

param prodcost {PROD} >= 0;
param invcost {PROD} >= 0;
param revenue {PROD,1..T} >= 0;
```

- python code with pandas

```
productDF = pd.read_excel("steelT.xlsx", index_col=0,
                          skip_footer=18)
revenue = pd.read_excel("steelT.xlsx", index_col=0,
                        skiprows=6, skip_footer=12)
revenue.index.name = 'Product'
market = pd.read_excel("steelT.xlsx", index_col=0,
                       skiprows=10, skip_footer=8)
market.index.name = 'Product'
avail = pd.read_excel("steelT.xlsx", index_col=0,
                      skiprows=16, usecols=1)

rmDF = pd.concat(
    [market.rename_axis("T", axis="columns")
     .stack().rename("market"),
     revenue.rename_axis("T", axis="columns")
     .stack().rename("revenue")],
    axis=1)
```

Decision Variable Declaration

- AMPL

```
var Make {PROD,1..T} >= 0;
var Inv {PROD,0..T} >= 0;
var Sell {p in PROD, t in 1..T} >= 0, <= market[p,t];
```

- pandas

```
from docplex.mp.model import Model
model = Model(name='steelT')
```

13.steelT.ipynb

```
Make = pd.Series(model.continuous_var_list(rmDF.index, name='Make'), index=rmDF.index,
                 name="Make")
Sell = pd.Series(model.continuous_var_list(rmDF.index, ub=list(rmDF.market.values),
                 name="Sell"), index=rmDF.index, name="Sell")
invindex = rmDF.index.union(
          pd.MultiIndex.from_product([rmDF.index.get_level_values("Product"), [0]],
                                     names=['Product','T']))
Inv = pd.Series(model.continuous_var_list(invindex, name='Inv'), index=invindex,
                name="Inv")
```

Note that we had to create an Index to contain time period 0

Decision Variables as a DataFrame

Product	T	Make	Sell	Inv
bands	0	NaN	NaN	Inv_('bands', 0)
	1	Make_('bands', 1)	Sell_('bands', 1)	Inv_('bands', 1)
	2	Make_('bands', 2)	Sell_('bands', 2)	Inv_('bands', 2)
	3	Make_('bands', 3)	Sell_('bands', 3)	Inv_('bands', 3)
	4	Make_('bands', 4)	Sell_('bands', 4)	Inv_('bands', 4)
coils	0	NaN	NaN	Inv_('coils', 0)
	1	Make_('coils', 1)	Sell_('coils', 1)	Inv_('coils', 1)
	2	Make_('coils', 2)	Sell_('coils', 2)	Inv_('coils', 2)
	3	Make_('coils', 3)	Sell_('coils', 3)	Inv_('coils', 3)
	4	Make_('coils', 4)	Sell_('coils', 4)	Inv_('coils', 4)

Objective Function

- AMPL

```
maximize Total_Profit:
   sum {p in PROD, t in 1..T} (revenue[p,t]*Sell[p,t] -
      prodcost[p]*Make[p,t] - invcost[p]*Inv[p,t]);
```

- python

```
model.maximize(model.sum(
         rmDF.revenue*Sell
        -productDF.prodcost*Make
        -productDF.invcost*Inv[rmDF.index]))
```

Time constraints

- AMPL

```
subject to Time {t in 1..T}:
   sum {p in PROD} (1/rate[p]) * Make[p,t] <= avail[t];
```

- python (using OptiPandas library)

```
opd.forall(model, "T",
          (opd.sum("Product", Make/productDF.rate) <= avail), "Time")
```

Initial Inventory/Balance constraint

- AMPL

```
subject to Init_Inv {p in PROD}:  Inv[p,0] = inv0[p];
```

- python (using OptiPandas library)

```
idx = pd.IndexSlice
Inv.sort_index(inplace=True)

opd.forall(model, "Product",
          (Inv[idx[:,0]] == productDF.inv0), "Init_Inv")
```

Balance Constraints

- AMPL

```
subject to Balance {p in PROD, t in 1..T}:
   Make[p,t] + Inv[p,t-1] = Sell[p,t] + Inv[p,t];
```

- python

```
opd.forall(model, ["Product", "T"],
          (Make + Inv.groupby('Product').shift(1)[rmDF.index]
           == Sell + Inv[rmDF.index]),
          "Balance")
```

What does shift(1) do?

```
Inv.groupby('Product').shift(1)
```

Product	T	
bands	0	NaN
	1	Inv_('bands', 0)
	2	Inv_('bands', 1)
	3	Inv_('bands', 2)
	4	Inv_('bands', 3)
coils	0	NaN
	1	Inv_('coils', 0)
	2	Inv_('coils', 1)
	3	Inv_('coils', 2)
	4	Inv_('coils', 3)

Solve and Obtain the Solution

```
model.solve()
soln = pd.DataFrame({'Make' : [x.solution_value for x in Make],
                     'Sell' : [x.solution_value for x in Sell],
                     'Inv' : [x.solution_value for x in Inv[rmDF.index]]},
                     index = rmDF.index)
```

		Inv	Make	Sell
Product	**T**			
bands	**1**	0.0	5990.0	6000.0
	2	0.0	6000.0	6000.0
	3	0.0	2040.0	2040.0
	4	0.0	2800.0	2800.0
coils	**1**	540.0	1967.0	1427.0
	2	0.0	1960.0	2500.0
	3	0.0	3500.0	3500.0
	4	0.0	4200.0	4200.0

A More Complex Example

Our Example Problem

- Assign hours of work of people to sites per day of week
- Each person can work
  - At most a specified number of hours per day
  - At most a specified number of hours per week
  - At most a specified number of days per week
- Each site needs a set of people assigned for a number of hours per day
- When a person is assigned to a site
  - They must be assigned for at least an hour, in half-hour increments
  - They must be assigned for at least 2 days in the week
- There is a preference score for each person/site pair that is eligible to be assigned
  - Maximize the preferences
- There are not enough people available to cover all the site requirements, so maximize the coverage as best as possible

Data: Sites.csv

```
id,num_hours
TWD0000000357BVYIM,12.0
UXE0000000358CWZJN,179.0
VYF0000000359DXAKO,171.0
WZG0000000360EYBLP,11.0
XAH0000000361FZCMQ,10.0
...
```

Data: People.csv

```
id,total_hrs_per_week,days_per_week
ADK0000000000ICFPT,6,3
BEL0000000001JDGQU,12,3
CFM0000000002KEHRV,10,7
DGN0000000003LFISW,28,6
EHO0000000004MGJTX,14,1
FIP0000000005NHKUY,25,3
GJQ0000000006OILVZ,20,4
...
```

Data: peoplehoursperday.csv

```
id,day,hours
GJQ0000000266OILVZ,Fr,2
UXE0000000150CWZJN,We,4
FIP0000000161NHKUY,Su,0
ILS0000000138QKNXB,We,0
SVC0000000278AUXHL,Tu,7
JMT0000000139RLOYC,Tu,4
PSZ0000000223XRUEI,We,8
...
```

Data: sitehoursperday.csv

```
id,day,hours
ADK000000364ICFPT,Mo,8.0
ADK000000364ICFPT,Tu,8.0
ADK000000364ICFPT,We,8.0
ADK000000364ICFPT,Th,8.0
ADK000000364ICFPT,Fr,4.0
ADK000000390ICFPT,Mo,8.0
ADK000000390ICFPT,Tu,8.0
ADK000000390ICFPT,We,8.0
ADK000000416ICFPT,Tu,2.5
ADK000000416ICFPT,We,2.5
ADK000000442ICFPT,Mo,7.0
...
```

Data: prefs.csv

```
person_id,site_id,pref
MPW0000000272UORBF,ADK0000000364ICFPT,64
ORY0000000196WQTDH,ADK0000000364ICFPT,32
QTA0000000146YSVFJ,ADK0000000364ICFPT,97
TWD0000000123BVYIM,ADK0000000364ICFPT,57
ADK0000000000ICFPT,ADK0000000390ICFPT,35
ADK0000000026ICFPT,ADK0000000390ICFPT,95
ADK0000000052ICFPT,ADK0000000390ICFPT,97
ADK0000000130ICFPT,ADK0000000390ICFPT,49
ADK0000000156ICFPT,ADK0000000390ICFPT,45
ADK0000000234ICFPT,ADK0000000390ICFPT,83
ADK0000000260ICFPT,ADK0000000390ICFPT,40
...
```

OPL Declarations to Read In Data

```
setof(string) sites = ...;
float num_hours_per_site[sites] = ...;
```

```
tuple tPeopleData {
    float total_hrs_per_week;
    float days_per_week;
}
setof(string) people = ...;
tPeopleData peopleData[people] = ...;
```

```
tuple tIdDayIndex {
    string id;
    string day;
};
setof(tIdDayIndex) siteHoursPerDayIndex = ...;
float hours_per_site_day[siteHoursPerDayIndex] = ...;
```

```
setof(tIdDayIndex) peopleHoursPerDayIndex = ...;
float hours_per_people_day[peopleHoursPerDayIndex] = ...;
```

```
tuple tPeopleSiteIndex {
    string person_id;
    string site_id;
};
setof(tPeopleSiteIndex) prefIndex= ...;
float pref[prefIndex] = ...;
```

Each declaration includes declaring an index set and an array of data indexed on that set

OPL Data File to Read in Data

```
SheetConnection siteSheet("sites.csv");
SheetConnection peopleSheet("people.csv");
SheetConnection siteHoursPerDaySheet("sitehoursperday.csv");
SheetConnection peopleHoursPerDaySheet("peoplehoursperday.csv");
SheetConnection prefSheet("prefs.csv");
```

```
sites from SheetRead(siteSheet, "A2:A549");
num_hours_per_site from SheetRead(siteSheet, "B2:B549");

people from SheetRead(peopleSheet, "A2:A358");
peopleData from SheetRead(peopleSheet, "B2:C358");

siteHoursPerDayIndex from SheetRead(siteHoursPerDaySheet, "A2:B1705");
hours_per_site_day from SheetRead(siteHoursPerDaySheet, "C2:C1705");

peopleHoursPerDayIndex from SheetRead(peopleHoursPerDaySheet, "A2:B2500");
hours_per_people_day from SheetRead(peopleHoursPerDaySheet, "C2:C2500");

prefIndex from SheetRead(prefSheet, "A2:B33301");
pref from SheetRead(prefSheet, "C2:C33301");
```

Read statements needed for each index set, and for each data array.
Precise locations in Excel worksheet are required.

Python (pandas) to read in data

```
import pandas as pd

sites = pd.read_csv("sites.csv", index_col=0)
sites.index.name = 'site_id'

people = pd.read_csv("people.csv", index_col=0)
people.index.name = 'person_id'

peoplehoursperday = pd.read_csv("peoplehoursperday.csv", index_col=[0,1])
peoplehoursperday.index.names = ['person_id', 'day']

sitehoursperday = pd.read_csv("sitehoursperday.csv", index_col=[0,1])
sitehoursperday.index.names = ['site_id', 'day']

prefs = pd.read_csv("prefs.csv", index_col=[0,1])
```

Do people and peoplehoursperday correspond?

OPL

```
setof(string) peopleInPeopleHoursPerDay =
      { p | <p, d> in peopleHoursPerDayIndex };
assert (card(people) == card(peopleInPeopleHoursPerDay));
assert forall (p in people) (p in peopleInPeopleHoursPerDay);
execute {
    writeln("PeopleHoursPerDay and People refer to same people.
Assertion succeeded.");
    writeln("Number of people = ", Opl.card(people));
};
```

pandas

```
uniquePeople =
    set(people.index.get_level_values('person_id'))

print ("PeopleHoursPerDay and People refer to same people? ",
   set(peoplehoursperday.index.get_level_values('person_id'))
         == uniquePeople)

print ("Number of people ", len(people))
```

Check that the people in the prefs table are in the people table

OPL

```
setof(string) peopleInPrefs = { p | <p,s> in prefIndex };

setof(string) peopleToPeoplePrefs = people diff peopleInPrefs;
assert (card(peopleToPeoplePrefs) == card(people) - card(peopleInPrefs));

execute {
    writeln("People in prefs and People refer to same people? ",
     (Opl.card(people) == Opl.card(peopleInPrefs)) &&
     (Opl.card(peopleToPeoplePrefs) == 0));

    writeln("Number of people in prefs = ", Opl.card(peopleInPrefs));
    writeln("People in prefs are in people table? ",
       Opl.card(peopleToPeoplePrefs) == Opl.card(people) - Opl.card(peopleInPrefs));
}
```

pandas

```
peopleInPrefs = set(prefs.index.get_level_values('person_id'))

print ("People in Prefs and People refer to same people? ",
       (len(uniquePeople) == len(peopleInPrefs)) and
       (uniquePeople == peopleInPrefs))

print ("Number of people in prefs = ", len(peopleInPrefs))

print ("People in prefs are in people table? ",
       peopleInPrefs.issubset(uniquePeople))
```

Output

```
PeopleHoursPerDay and People refer to same people?  True
Number of people  357

People in Prefs and People refer to same people?  False
Number of people in prefs =  338

People in prefs are in people table?  True
```

So we can conclude that there are people who have no sites they can be assigned to

There are additional data checks (omitted for brevity)

Merging/Joining

peoplehoursperday		hours
id	**day**	
ADK0000000000ICFPT	**Mo**	0
BEL0000000001JDGQU	**Mo**	8
CFM0000000002KEHRV	**Mo**	0
DGN0000000003LFISW	**Mo**	0
EHO0000000004MGJTX	**Mo**	0

prefs table		pref
person_id	**site_id**	
MPW0000000272UORBF	**ADK0000000364ICFPT**	64
ORY0000000196WQTDH	**ADK0000000364ICFPT**	32
QTA0000000146YSVFJ	**ADK0000000364ICFPT**	97
TWD0000000123BVYIM	**ADK0000000364ICFPT**	57
ADK0000000000ICFPT	**ADK0000000390ICFPT**	35

sitehoursperday		hours
id	**day**	
	Mo	8.0
	Tu	8.0
ADK0000000364ICFPT	**We**	8.0
	Th	8.0
	Fr	4.0

	person_id	site_id	pref	day	daily_hours
0	MPW0000000272UORBF	ADK0000000364ICFPT	64	Mo	8.0
1	MPW0000000272UORBF	ADK0000000364ICFPT	64	Tu	8.0
2	MPW0000000272UORBF	ADK0000000364ICFPT	64	We	8.0
3	MPW0000000272UORBF	ADK0000000364ICFPT	64	Th	4.0
4	MPW0000000272UORBF	ADK0000000364ICFPT	64	Fr	4.0
9	ORY0000000196WQTDH	ADK0000000364ICFPT	32	We	4.0
10	ORY0000000196WQTDH	ADK0000000364ICFPT	32	Th	4.0
11	ORY0000000196WQTDH	ADK0000000364ICFPT	32	Fr	4.0
14	QTA0000000146YSVFJ	ADK0000000364ICFPT	97	Mo	6.0
15	QTA0000000146YSVFJ	ADK0000000364ICFPT	97	Tu	6.0
16	QTA0000000146YSVFJ	ADK0000000364ICFPT	97	We	6.0
17	QTA0000000146YSVFJ	ADK0000000364ICFPT	97	Th	6.0
21	TWD0000000123BVYIM	ADK0000000364ICFPT	57	Mo	6.0
22	TWD0000000123BVYIM	ADK0000000364ICFPT	57	Tu	6.0

Merging: OPL vs. pandas

OPL

```
setof(tIdDayIndex) prefPeopleHoursByDayIndexAll =
    { <p,d> | p in peopleInPrefs, <p,d> in peopleHoursPerDayIndex };
setof(tIdDayIndex) prefSitesHoursByDayIndexAll =
    { <s,d> | s in sitesInPrefs, <s,d> in siteHoursPerDayIndex };

tuple tPersonSiteDay {
    string person_id;
    string site_id;
    string day;
};
setof(tPersonSiteDay) prefsPersonSiteDayIndexAll =
    { <p,s,d> | <p,s> in prefIndex,
                <p,d> in prefPeopleHoursByDayIndexAll,
                <s,d> in prefSitesHoursByDayIndexAll };

float hoursPerPersonSiteDayAll[<p,s,d> in prefsPersonSiteDayIndexAll] =
    minl(hours_per_people_day[<p,d>], hours_per_site_day[<s,d>]);

setof(tPersonSiteDay) prefsPersonSiteDayIndex =
    { <p,s,d> | <p,s,d> in prefsPersonSiteDayIndexAll : hoursPerPersonSiteDayAll[<p,s,d>] > 0};
```

pandas

```
prefswithdaily = (
    pd.merge(prefs.reset_index(), peoplehoursperday.reset_index(),
             on='person_id', how='left')
    .rename(columns={'hours' : 'person_hours'})
    .merge(sitehoursperday.reset_index(), on=['site_id','day'], how='inner')
    .rename(columns={'hours' : 'site_hours'})
    )
prefswithdaily['daily_hours'] = prefswithdaily[['person_hours','site_hours']].min(axis=1)
prefswithdaily = prefswithdaily.drop(['person_hours','site_hours'],axis=1)
                 [prefswithdaily.daily_hours > 0]
```

Determining Eligible pairs, people, sites

OPL

```
setof(tPersonSiteDay) allVarsIndex = prefsPersonSiteDayIndex;

setof(tPeopleSiteIndex) eligiblePairs = { <p,s> | <p,s,d> in allVarsIndex };

setof(string) eligiblePeople = { p | <p,s> in eligiblePairs };

setof(string) eligibleSites = { s | <p,s> in eligiblePairs };
```

pandas

```
allvars = prefswithdaily.set_index(['person_id','site_id','day'])

eligiblePairs = set((p,s) for (p,s,d) in allvars.index)

eligiblePeople = set(p for p,s in eligiblePairs)

eligibleSites = set(s for p,s in eligiblePairs)
```

Decision Variables

OPL

```
int blocksPerPersonSiteDay[<p,s,d> in prefsPersonSiteDayIndex] =
      ftoi(round(2.0*hoursPerPersonSiteDayAll[<p,s,d>]));

dvar int+ h[<p,s,d> in allVarsIndex] in 0..blocksPerPersonSiteDay[<p,s,d>];
dvar int+ pairday[<p,s,d> in allVarsIndex] in 0..1;
dvar int+ z[eligiblePairs] in 0..1;
dvar float+ shortage[eligibleSites];
dvar float+ excess[eligibleSites];
```

pandas

```
from gurobipy import *
m=Model("sched")

h = pd.Series(m.addVars(allvars.index ub=2*allvars.daily_hours.values,
                        vtype=GRB.INTEGER, name="h").values(),
              index=allvars.index, name='h')
pairday = pd.Series(m.addVars(allvars.index, vtype=GRB.BINARY,
                              name='pairday').values(),
                    index=allvars.index, name='pairday')

z = pd.Series(m.addVars(eligiblePairs, vtype=GRB.BINARY, name="z").values(),
              index=pd.MultiIndex.from_tuples(eligiblePairs,
                                              names=['person_id','site_id']),
              name='z')

shortage = pd.Series(m.addVars(eligibleSites, vtype=GRB.CONTINUOUS,
                               name='shortage').values(),
                     index=pd.Index(eligibleSites, name='site_id'), name='shortage')

excess = pd.Series(m.addVars(eligibleSites, vtype=GRB.CONTINUOUS,
                             name='excess').values(),
                   index=pd.Index(eligibleSites, name='site_id'), name='excess')
```

Objective Function (Hierarchical)

OPL

```
minimize 1.0*totalShortage + 0.0*totalExcess + 0.0*totalPrefs;

totalShortage == sum(s in eligibleSites) shortage[s];
totalExcess == sum(s in eligibleSites) excess[s];
totalPrefs == sum(p in eligiblePairs) pref[p]*z[p];

/* Script */
thisOplModel.generate();
cplex.solve();
thisOplModel.postProcess();
var objVal = cplex.getObjValue();

cplex.setObjCoef(thisOplModel.totalShortage, 0);
cplex.setObjCoef(thisOplModel.totalExcess, 1);
thisOplModel.shortageBound.UB = objVal + 0.25;
cplex.solve();
objVal = cplex.getObjValue();
thisOplModel.excessBound.UB = objVal + 0.25;
cplex.setObjCoef(thisOplModel.totalExcess, 0);
cplex.setObjCoef(thisOplModel.totalPrefs, -1); // To maximize
cplex.solve();
```

pandas

```
m.setObjective(quicksum(shortage), sense=GRB.MINIMIZE)
m.optimize()
m.addConstr(quicksum(shortage) <= m.ObjVal+0.25, name='shortageBound')
m.setObjective(quicksum(excess))
m.optimize()
m.addConstr(quicksum(siteSlackVars.excess) <= m.ObjVal+0.25, name='excessBound')

conDF = zvars.merge(prefs, left_index=True, right_index=True, how='left')
m.setObjective(quicksum(prefs.pref[z.index]*z), sense=GRB.MAXIMIZE)
m.optimize()
```

Constraint: Limit Hours Per Person Per Day

$$\sum_{(p,s,d)} h[p,s,d] \leq 2 * people\_hours\_per\_day[p,d]\ ,\ \forall\ (p,d)$$

OPL

```
forall (<p,d> in prefPeopleHoursByDayIndex) {
PersonHoursPerDay[<p,d>]:
    sum(<p,s,d> in allVarsIndex) h[<p,s,d>] <= 2*hours_per_people_day[<p,d>];
}
```

Gurobi

```
phpdict = { k : v['hours']
            for k,v in peoplehoursperday.to_dict(orient='index').items() }

m.addConstrs((h.sum(person_id,'*',day) <= 2*phpdict[person_id,day]
             for (person_id, day) in
                  set((person_id,day) for (person_id,site_id,day) in h.keys())),
             name='PersonHoursPerDay')
```

Pandas

```
opd.forall(m, ["person_id", "day"],
    (opd.sum('site_id', h) <= 2*peoplehoursperday.hours), "PersonHoursPerDay")
```

Constraint: Limit Hours Per Site Per Day

OPL

```
forall (<s,d> in prefSitesHoursByDayIndex) {
SiteHoursPerDay[<s,d>]:
    sum(<p,s,d> in allVarsIndex) h[<p,s,d>] <= 2*hours_per_site_day[<s,d>];
}
```

Pandas

```
opd.forall(m, ["site_id", "day"],
    (opd.sum('person_id', h) <= 2*sitehoursperday.hours), "SiteHoursPerDay")
```

Constraint: Limit Hours Per Week

OPL

```
forall (p in eligiblePeople) {
TotalHoursPerWeek[p]:
    sum(<p,s,d> in allVarsIndex) h[<p,s,d>] <= 2*peopleData[p].total_hrs_per_week;
}
```

Pandas

```
opd.forall(m, "person_id",
    (opd.sum(['site_id','day'], h) <= 2*people.total_hrs_per_week),
    "TotalHoursPerWeek")
```

Constraint: Must be assigned for at least an hour

O
P
L

```
forall (<p,s,d> in allVarsIndex) {
PairDayUp[<p,s,d>]:
    h[<p,s,d>] <= blocksPerPersonSiteDay[<p,s,d>]*pairday[<p,s,d>];
PairDayLow[<p,s,d>]:
    2*pairday[<p,s,d>] <= h[<p,s,d>];
}
```

person_id	site_id	day	pref	daily_hours	h	pairday
MPW0000000272UORBF	**ADK0000000364ICFPT**	**Mo**	64	8.0	<gurobi.Var h[MPW0000000272UORBF, ADK0000000364ICFPT,Mo] >	<gurobi.Var pairday[MPW0000000272UORBF, ADK0000000364ICFPT,Mo]>
		Tu	64	8.0	<gurobi.Var h[MPW0000000272UORBF, ADK0000000364ICFPT,Tu]>	<gurobi.Var pairday[MPW0000000272UORBF, ADK0000000364ICFPT,Tu]>

P
a
n
d
a
s

```
opd.forall(m, ["person_id", "site_id", "day"],
          (h <= 2*allvars.daily_hours*pairday), "PairDayUp")
opd.forall(m, ["person_id", "site_id", "day"],
          (2*pairday <= h), "PairDayLow")
```

Constraint: Limit on Total Days Per Week

O
P
L

```
forall (p in eligiblePeople) {
TotalDaysPerWeek[p]:
    sum(<p,s,d> in allVarsIndex) pairday[<p,s,d>] <= peopleData[p].days_per_week;
}
```

P
a
n
d
a
s

```
opd.forall(m, "person_id‟,
          (opd.sum(["site_id", "day"], pairday) <= people.days_per_week),
          "TotalDaysPerWeek")
```

Constraint: If assigned to a site, must be for at least 2 days

OPL

```
forall (<p,s> in eligiblePairs) {
ZVarUp[<p,s>]:
    2*z[<p,s>] <= sum(<p,s,d> in allVarsIndex) pairday[<p,s,d>];
ZVarLow[<p,s>]:
    sum(<p,s,d> in allVarsIndex) pairday[<p,s,d>] <=
           card({<p,s,d> | <p,s,d> in allVarsIndex})*z[<p,s>];
}
```

Compute the sum as an object. Use number of variables in sum to compute big-M

Pandas

```
sumpairday = opd.sum("day", pairday)

opd.forall(m, ["person_id", "site_id"],
           (2*z <= sumpairday), 'ZVarUp')

sumpairdayBigM = sumpairday.apply(lambda v : v.size())

opd.forall(m, ["person_id", "site_id"],
          (sumpairday <= sumpairdayBigM*z), 'ZVarLow')
```

Constraint: Track Hours Assigned to Site

OPL

```
forall (s in eligibleSites) {
SiteHours[s]:
    sum(<p,s,d> in allVarsIndex) h[<p,s,d>] + shortage[s] - excess[s]
            == 2*num_hours_per_site[s];
}
```

Pandas

```
opd.forall(m, ["site_id"],
    (opd.sum(["person_id", "day"], h) + shortage - excess == 2*sites.num_hours),
          "SiteHours")
```

Solving

```
m.setParam(GRB.Param.TimeLimit, 300)
m.setParam(GRB.Param.CutPasses, 1)
m.optimize()

m.addConstr(quicksum(shortage) <= m.ObjVal+0.25, name='shortageBound')
m.setObjective(quicksum(excess))

resetParams()
m.setParam(GRB.Param.TimeLimit, 180)
m.setParam(GRB.Param.Presolve, 0)
m.setParam(GRB.Param.Cuts, 0)
m.optimize()

m.addConstr(quicksum(.excess) <= m.ObjVal+0.25, name='excessBound')
conDF = zvars.merge(prefs, left_index=True, right_index=True, how='left')
m.setObjective(quicksum(prefs.pref[z.index]*z), sense=GRB.MAXIMIZE)

resetParams()
m.setParam(GRB.Param.MIPGap, 0.01) # Set gap to 1%
m.setParam(GRB.Param.Cuts, 0)
m.setParam(GRB.Param.SubMIPNodes, 1500)
m.setParam(GRB.Param.TimeLimit, 180)
m.optimize()
```

Solution

```
hvals = pd.Series([h.x for h in h], index=h.index)
pairings = hvals[hvals>0]
```

			hours
person_id	site_id	day	
ORY0000000196WQTDH	ADK0000000364ICFPT	Th	4.0
		Fr	4.0
QTA0000000146YSVFJ	ADK0000000364ICFPT	Mo	2.0
		Tu	12.0
		We	12.0
TWD0000000123BVYIM	ADK0000000364ICFPT	Mo	12.0
		Th	12.0
ADK0000000130ICFPT	ADK0000000390ICFPT	Mo	16.0
		Tu	2.0
BEL0000000235JDGQU	ADK0000000390ICFPT	Tu	14.0

Conclusion

- pandas and Python provide a powerful combination for working with data used to build optimization models
  - There are other powerful features in pandas for analyzing and organizing data
    - Reshaping, Grouping, Functions on data, plotting
- Using pandas, you can write efficient code for building optimization models
  - With OptiPandas from Princeton Consultants, write vectors of constraints
- Think Different!
  - Data FIRST
  - Model SECOND