gurobipy-pandas

Building optimization models from pandas dataframes

Robert Luce, Principal Developer

github.com/Gurobi/gurobipy-pandas#webinar

gurobipy

- Python API for the Gurobi Optimizer

pandas

- Flexible data analysis and manipulation tool for Python
- Open source at github.com/pandas-dev/pandas
- NumFOCUS sponsored project
- Documentation on pydata.org
- Provides DataFrames for Python; plus I/O, analysis, plotting & more
- Standard package for analytics projects in Python

gurobipy-pandas

- Linking package between `gurobipy` and `pandas`
- Open source at github.com/Gurobi/gurobipy-pandas
- Developed by several enthusiasts at Gurobi, with input from Princeton Consultants
- Documentation on readthedocs
- Allows users to build Gurobi models from Pandas data in a readable way

Design Principles

- Pandas DataFrames and Series already define *data* over indexes

In [1]:
```
import pandas as pd
import numpy as np
np.random.seed(0)

df = pd.DataFrame(
    index=pd.RangeIndex(4, name="i"),
    columns=["a", "b"],
    data=np.random.random((4, 2)).round(2)
)
df
```

Out[1]:

i	a	b
0	0.55	0.72
1	0.60	0.54
2	0.42	0.65
3	0.44	0.89

- We need a way to *define variables* and *build constraints* over the same indexes

In [3]:

```
# Create a gurobipy model
model = gp.Model()
```

In [4]:
```
i = pd.RangeIndex(5, name="i")
i
```

Out[4]:
```
RangeIndex(start=0, stop=5, step=1, name='i')
```

Creating variables

Using the free function `gppd.add_vars` :

- Creates one variable per entry in the index
- Returns a pandas `Series` of gurobipy `Var` objects
- Variable names are based on index values
- Variable attributes can be set

In [4]:
```
i = pd.RangeIndex(5, name="i")
i
```

Out[4]:
```
RangeIndex(start=0, stop=5, step=1, name='i')
```

In [5]:
```
x = gppd.add_vars(model, i, name="x", vtype="B")
x
```

Out[5]:
```
i
0    <gurobi.Var x[0]>
1    <gurobi.Var x[1]>
2    <gurobi.Var x[2]>
3    <gurobi.Var x[3]>
4    <gurobi.Var x[4]>
Name: x, dtype: object
```

Single indexes

Consider an index $i \in I$, some variables x_i, and some data c_i:

In [6]:
```
i = pd.RangeIndex(5, name="i")
i
```

Out[6]:
```
RangeIndex(start=0, stop=5, step=1, name='i')
```

In [7]:
```
x = gppd.add_vars(model, i, name="x")
x
```

Out[7]:
```
i
0    <gurobi.Var x[0]>
1    <gurobi.Var x[1]>
2    <gurobi.Var x[2]>
3    <gurobi.Var x[3]>
4    <gurobi.Var x[4]>
Name: x, dtype: object
```

In [8]:
```
c = pd.Series(index=i, name="c", data=np.arange(1, 6))
c
```

Out[8]:
```
i
0    1
1    2
2    3
```

```
3    4
4    5
Name: c, dtype: int64
```

Arithmetic with scalars

$$2x_i + 5 \quad \forall i \in I$$

- Produce a new series on the same index
- One linear expression per entry in the index

In [9]:
```
2*x + 5
```

Out[9]:
```
i
0    5.0 + 2.0 x[0]
1    5.0 + 2.0 x[1]
2    5.0 + 2.0 x[2]
3    5.0 + 2.0 x[3]
4    5.0 + 2.0 x[4]
Name: x, dtype: object
```

Summation

$$\sum_i x_i$$

- Produce a single linear expression
- Sums the whole series over the index

```
In [10]: x.sum()

Out[10]: <gurobi.LinExpr: x[0] + x[1] + x[2] + x[3] + x[4]>
```

Arithmetic with Series

$$c_i x_i \quad \forall i \in I$$

- Produce a new series on the same index
- Pointwise product for each entry in the index

In [11]:
```
c * x
```

Out[11]:
```
i
0        x[0]
1    2.0 x[1]
2    3.0 x[2]
3    4.0 x[3]
4    5.0 x[4]
dtype: object
```

Summing the result

$$\sum_{i \in I} c_i x_i$$

- Produce a single linear expression
- Take our pointwise product series, sum over the index

In [12]:
```
(c * x).sum()
```

Out[12]:
```
<gurobi.LinExpr: x[0] + 2.0 x[1] + 3.0 x[2] + 4.0 x[3] + 5.0 x
[4]>
```

Looking Familiar?

Hopefully!

Any operation you would do with data in pandas, you can do in the same way with data & variables.

Multi-Index

- Multi-indexes allow us to add dimensions for data and variables
- Start with an example DataFrame, representing the data p_{ij}

In [13]:
```
data = pd.DataFrame({
    "i": [0, 0, 1, 2, 2],
    "j": [1, 2, 0, 0, 1],
    "p": [0.1, 0.6, 1.2, 0.4, 0.9],
}).set_index(["i", "j"])
data
```

Out[13]:

		p
i	**j**	
0	**1**	0.1
	2	0.6
1	**0**	1.2
2	**0**	0.4
	1	0.9

Multi-Index

- Add corresponding variables y_{ij} as a Series:

Multi-Index

- Add corresponding variables y_{ij} as a Series:

In [14]:
```
y = gppd.add_vars(model, data, name="y")
y
```

Out[14]:
```
i  j
0  1    <gurobi.Var y[0,1]>
   2    <gurobi.Var y[0,2]>
1  0    <gurobi.Var y[1,0]>
2  0    <gurobi.Var y[2,0]>
   1    <gurobi.Var y[2,1]>
Name: y, dtype: object
```

Multi-Index

- Add corresponding variables y_{ij} as a Series:

In [14]:
```
y = gppd.add_vars(model, data, name="y")
y
```

Out[14]:
```
i  j
0  1    <gurobi.Var y[0,1]>
   2    <gurobi.Var y[0,2]>
1  0    <gurobi.Var y[1,0]>
2  0    <gurobi.Var y[2,0]>
   1    <gurobi.Var y[2,1]>
Name: y, dtype: object
```

- Note that we just pass the dataframe and get back a matching index
- 1:1 correspondence between data and variables over a given index

Grouped summation

$$\sum_{i \in I} y_{ij} \quad \forall j \in J$$

- For each j, sum y_{ij} terms over all corresponding valid i values
- Produces a Series of linear expressions, indexed by j

In [15]:
```
y
```

Out[15]:
```
i  j
0  1    <gurobi.Var y[0,1]>
   2    <gurobi.Var y[0,2]>
1  0    <gurobi.Var y[1,0]>
2  0    <gurobi.Var y[2,0]>
   1    <gurobi.Var y[2,1]>
Name: y, dtype: object
```

In [16]:
```
y.groupby("j").sum()
```

Out[16]:
```
j
0          y[1,0] + y[2,0]
1          y[0,1] + y[2,1]
2    <gurobi.Var y[0,2]>
Name: y, dtype: object
```

Finally...

$$\sum_{j \in J} c_j y_{ij} \quad \forall i \in I$$

- Use the series `c * y`
- Apply the same groupby-aggregate operation as before
- Result is a series indexed by i

```
In [17]: (c * y).groupby("i").sum()
```

```
Out[17]: i
         0              y[0,1] + y[0,2]
         1                   2.0 y[1,0]
         2    3.0 y[2,0] + 3.0 y[2,1]
         dtype: object
```

Pandas arithmetic

- Pandas aligns before applying arithmetic operators
    - see e.g. align and add
- This alignment is all performed by pandas
- Hence it follows pandas' defined behaviour:
    - Joining
    - Matching
    - Aligning
    - Broadcasting

Creating Constraints

- Indexes must align between two series
- Aim to build vectorized constraints (no manual iteration)

$$\sum_j c_j y_{ij} \leq b_i \quad \forall i \in I$$

In [18]:
```
(c * y).groupby("i").sum()
```

Out[18]:
```
i
0              y[0,1] + y[0,2]
1                   2.0 y[1,0]
2    3.0 y[2,0] + 3.0 y[2,1]
dtype: object
```

In [19]:
```
b = pd.Series(index=pd.RangeIndex(3, name="i"), data=[1, 2, 3])
b
```

Out[19]:
```
i
0    1
1    2
2    3
dtype: int64
```

- Use `gppd.add_constrs` free function
- Return a series of constraint handles

In [20]:
```
constraints = gppd.add_constrs(
    model,
    (c * y).groupby("i").sum(),  # left-hand side
    GRB.LESS_EQUAL,              # inequality (sense)
    b,                           # right-hand side
    name="constr",
)
constraints
```

Out[20]:
```
i
0    <gurobi.Constr constr[0]>
1    <gurobi.Constr constr[1]>
2    <gurobi.Constr constr[2]>
Name: constr, dtype: object
```

Setting the Objective

- Objectives are set from single expressions
- No `gurobipy-pandas` method here (no vectorized operations)

In [21]:
```
model.setObjective(y.sum(), sense=GRB.MAXIMIZE)
```

In [22]:
```
model.update()
model.getObjective()
```

Out[22]:
```
<gurobi.LinExpr: y[0,1] + y[0,2] + y[1,0] + y[2,0] + y[2,1]>
```

Extracting Solutions

- In `gurobipy`, solutions are retrieved from the `.X` attribute of a `Var`
- `gppd` *Series accessor* vectorizes this operation
- Works for any attributes (bounds, coefficients, RHS, etc)
- Returns a Series on the same index

In [23]:
```
model.optimize()
y.gppd.X  # Series accessor
```

Out[23]:
```
i  j
0  1    0.0
   2    1.0
1  0    1.0
2  0    0.0
   1    1.0
Name: y, dtype: float64
```

The Model

- Given a set of projects $i \in I$ and teams $j \in J$
- Project i requires w_i resources to complete, and each team j has capacity c_j
- If team j completes project i, we profit p_{ij}
- Goal: maximize the value of completed projects, respecting team capacities

The Data

- *Before* taking any modelling steps: prepare your data properly:
- Clearly define your model indexes, align dataframes to these indexes
- Keep data reading & cleaning separate from model building

In [27]:
```
projects = pd.read_csv(projects_csv, index_col="project")
projects.head(3)  # w_i
```

Out[27]:

project	resource
0	1.1
1	1.4
2	1.2

In [28]:
```
teams = pd.read_csv(teams_csv, index_col="team")
teams.head(3)  # c_j
```

Out[28]:

team	capacity
0	2.4
1	1.8
2	1.1

Sparsity

- Note that the model is not defined over all (i, j) pairs
- Not all teams can complete all projects
- There are 80 < 150 combinations (sparse!)
- Structure of the data matches the model

In [29]:
```
project_values = pd.read_csv(project_values_csv, index_col=["project",
project_values  # p_ij
```

Out[29]:

		profit
project	**team**	
0	**4**	0.4
1	**4**	1.3
2	**0**	1.7
	1	1.7
	2	1.7
...	**...**	...
28	**2**	1.0
	3	1.0
	4	1.0

		profit
project	team	
29	**3**	1.3

Clean data

- All indexes (i, j, ij) are correctly represented in the data
- There are no missing values
- Index alignment is correct
    - Every project in `project_values` has an entry in `projects`
    - Every team in `project_values` has an entry in `teams`

Define variables and objective

- Maximize the total value of completed projects

$$\max \quad \sum_{i \in I} \sum_{j \in J} p_{ij} x_{ij} \tag{4}$$

$$\text{s.t.} \quad x_{ij} \in \{0, 1\} \quad \forall (i, j) \tag{5}$$

In [30]:
```
model = gp.Model()
model.ModelSense = GRB.MAXIMIZE
x = gppd.add_vars(model, project_values, vtype=GRB.BINARY, obj="profit"
x.head()
```

Out[30]:
```
project  team
0        4       <gurobi.Var x[0,4]>
1        4       <gurobi.Var x[1,4]>
2        0       <gurobi.Var x[2,0]>
         1       <gurobi.Var x[2,1]>
         2       <gurobi.Var x[2,2]>
Name: x, dtype: object
```

Define variables and objective

- Maximize the total value of completed projects

$$\max \quad \sum_{i \in I} \sum_{j \in J} p_{ij} x_{ij} \tag{6}$$

$$\text{s.t.} \quad x_{ij} \in \{0, 1\} \quad \forall (i, j) \tag{7}$$

In [30]:
```
model = gp.Model()
model.ModelSense = GRB.MAXIMIZE
x = gppd.add_vars(model, project_values, vtype=GRB.BINARY, obj="profit"
x.head()
```

Out[30]:
```
project  team
0        4       <gurobi.Var x[0,4]>
1        4       <gurobi.Var x[1,4]>
2        0       <gurobi.Var x[2,0]>
         1       <gurobi.Var x[2,1]>
         2       <gurobi.Var x[2,2]>
Name: x, dtype: object
```

In [31]:
```
model.getObjective()
```

Out[31]:
```
<gurobi.LinExpr: 0.4 x[0,4] + 1.3 x[1,4] + 1.7 x[2,0] + 1.7 x
[2,1] + 1.7 x[2,2] + 1.7 x[2,3] + 1.7 x[2,4] + 1.3 x[3,4] + 1.3
x[4,0] + 1.3 x[4,1] + 1.3 x[4,2] + 1.3 x[4,3] + 1.3 x[4,4] + 1.
```

8 x[5,0] + 1.8 x[5,1] + 1.8 x[5,2] + 1.8 x[5,3] + 1.8 x[5,4] +
1.2 x[6,0] + 1.2 x[6,1] + 1.2 x[6,2] + 1.2 x[6,3] + 1.2 x[6,4]
+ 0.9 x[7,3] + 0.9 x[7,4] + x[8,3] + x[8,4] + 1.2 x[9,4] + 0.8
x[10,0] + 0.8 x[10,1] + 0.8 x[10,2] + 0.8 x[10,3] + 0.8 x[10,4]
+ 1.3 x[11,0] + 1.3 x[11,1] + 1.3 x[11,2] + 1.3 x[11,3] + 1.3 x
[11,4] + 0.8 x[12,3] + 0.8 x[12,4] + 1.5 x[13,0] + 1.5 x[13,1]
+ 1.5 x[13,2] + 1.5 x[13,3] + 1.5 x[13,4] + 1.7 x[14,3] + 1.7 x
[14,4] + 1.3 x[15,4] + 0.3 x[16,4] + 1.2 x[17,0] + 1.2 x[17,1]
+ 1.2 x[17,2] + 1.2 x[17,3] + 1.2 x[17,4] + 1.3 x[18,3] + 1.3 x
[18,4] + 1.8 x[19,3] + 1.8 x[19,4] + 1.6 x[20,3] + 1.6 x[20,4]
+ 1.1 x[21,3] + 1.1 x[21,4] + 0.4 x[22,4] + x[23,4] + 0.3 x[24,
4] + x[25,0] + x[25,1] + x[25,2] + x[25,3] + x[25,4] + 1.8 x[2
6,4] + 0.8 x[27,3] + 0.8 x[27,4] + x[28,0] + x[28,1] + x[28,2]
+ x[28,3] + x[28,4] + 1.3 x[29,3] + 1.3 x[29,4]>

Capacity constraint

- Assigned projects are limited by team capacity

$$\sum_{i \in I} w_i x_{ij} \leq c_j \quad \forall j \in J$$

In [32]:
```
capacity_constraints = gppd.add_constrs(
    model,
    (
        (projects["resource"] * x)
        .groupby("team").sum()
    ),
    GRB.LESS_EQUAL,
    teams["capacity"],
    name='capacity',
)
capacity_constraints
```

Out[32]:
```
team
0    <gurobi.Constr capacity[0]>
1    <gurobi.Constr capacity[1]>
2    <gurobi.Constr capacity[2]>
3    <gurobi.Constr capacity[3]>
4    <gurobi.Constr capacity[4]>
Name: capacity, dtype: object
```

Allocate once

- Each project is allocated at most once

$$\sum_{j \in J} x_{ij} \leq 1 \quad \forall i \in I$$

In [33]:
```
allocate_once = gppd.add_constrs(
    model, x.groupby('project').sum(),
    GRB.LESS_EQUAL, 1.0, name="allocate_once",
)
allocate_once.head()
```

Out[33]:
```
project
0    <gurobi.Constr allocate_once[0]>
1    <gurobi.Constr allocate_once[1]>
2    <gurobi.Constr allocate_once[2]>
3    <gurobi.Constr allocate_once[3]>
4    <gurobi.Constr allocate_once[4]>
Name: allocate_once, dtype: object
```

Solutions

- Optimize the model
- Get back solution values as a series on our original index

In [34]:
```
model.optimize()
x.gppd.X.head()
```

Out[34]:
```
project  team
0        4       0.0
1        4       0.0
2        0      -0.0
         1       1.0
         2       0.0
Name: x, dtype: float64
```

Solutions

- Optimize the model
- Get back solution values as a series on our original index

In [34]:
```
model.optimize()
x.gppd.X.head()
```

Out[34]:
```
 project  team
0         4       0.0
1         4       0.0
2         0      -0.0
          1       1.0
          2       0.0
Name: x, dtype: float64
```

In [35]:
```
(
    x.gppd.X.to_frame()
    .query("x >= 0.9").reset_index()
    .groupby("team").agg({"project": list})
)
```

Out[35]:

team	project
0	[4, 5]

	project
team	
1	[2]
2	[11]