Monte Carlo Business Case Analysis using pandas

Demonstration

In [1]:
```
import numpy as np
from pandas import Series, DataFrame
import pandas as pd
```

The inputs are:

- revenues
  - volume = number of trees × fruit per tree
  - price/volume
- costs
  - fixed costs
  - variable costs

We will:

- Read & combine inputs
- Simulate (draw random values)
- Calculate revenue and costs
- Report results

Read inputs

In [2]:
```
num_trees = pd.read_csv('data/numTrees.csv', skipinitialspace=True)
num_trees
```

Out[2]:

	farm	fruit	variety	num trees
0	Farm A	apple	Granny Smith	100
1	Farm A	apple	Jonathon	200
2	Farm A	pear	Nashi	50
3	Farm B	apple	Granny Smith	200
4	Farm B	apple	Jonathon	400

5 rows × 4 columns

In [7]:
```
fruit_per_tree = pd.read_csv('data/fruitPerTree2.csv', skipinitialspace=True)
fruit_per_tree
```

Out[7]:

	fruit	variety	mean	sd
0	apple	Granny Smith	200	30
1	apple	Jonathon	150	22
2	apple	Red Delicious	150	20
3	pear	Nashi	50	8

4 rows × 4 columns

Combine inputs

In [14]:
```
fruit = pd.merge(num_trees, fruit_per_tree)
fruit
# note broadcasting just happened
```

Out[14]:

	farm	fruit	variety	num trees	mean	sd
0	Farm A	apple	Granny Smith	100	200	30
1	Farm B	apple	Granny Smith	200	200	30
2	Farm A	apple	Jonathon	200	150	22
3	Farm B	apple	Jonathon	400	150	22
4	Farm A	pear	Nashi	50	50	8

5 rows × 6 columns

In [12]:
```
# also note the Red Delicious line is missing; include it explicitly below
pd.merge(num_trees, fruit_per_tree, how='outer')
```

Out[12]:

	farm	fruit	variety	num trees	mean	sd
0	Farm A	apple	Granny Smith	100	200	30
1	Farm B	apple	Granny Smith	200	200	30
2	Farm A	apple	Jonathon	200	150	22
3	Farm B	apple	Jonathon	400	150	22
4	Farm A	pear	Nashi	50	50	8
5	NaN	apple	Red Delicious	NaN	150	20

6 rows × 6 columns

Simulate (draw random samples)

We want to expand out every line in this table into another dimension.
In the `dataframe` paradigm, we add another column, and repeat the indices.
We can achieve this by using `groupby` and `apply`

In [15]:
```
fruit
```

Out[15]:

	farm	fruit	variety	num trees	mean	sd
0	Farm A	apple	Granny Smith	100	200	30
1	Farm B	apple	Granny Smith	200	200	30
2	Farm A	apple	Jonathon	200	150	22
3	Farm B	apple	Jonathon	400	150	22
4	Farm A	pear	Nashi	50	50	8

5 rows × 6 columns

In [16]:
```
fruit.groupby(['farm']).count()
```

Out[16]:

	farm	fruit	variety	num trees	mean	sd
farm						
Farm A	3	3	3	3	3	3
Farm B	2	2	2	2	2	2

2 rows × 6 columns

In [17]:
```
fruit.groupby(['farm','fruit']).count()
```

Out[17]:

		farm	fruit	variety	num trees	mean	sd
farm	**fruit**						
Farm A	**apple**	2	2	2	2	2	2
	pear	1	1	1	1	1	1
Farm B	**apple**	2	2	2	2	2	2

3 rows × 6 columns

In [18]:
```
def test_func(grp):
    return grp.count()
fruit.groupby(['farm','fruit']).apply(test_func)
```

Out[18]:

		farm	fruit	variety	num trees	mean	sd
farm	**fruit**						
Farm A	**apple**	2	2	2	2	2	2
	pear	1	1	1	1	1	1
Farm B	**apple**	2	2	2	2	2	2

3 rows × 6 columns

In [19]:
```
def sim(grp):
    return Series({'random': np.random.normal(), 'count': grp.count().mean()})
fruit.groupby(['farm','fruit']).apply(sim)
```

Out[19]:

		count	random
farm	**fruit**		
Farm A	**apple**	2	-0.712683
	pear	1	-0.463357
Farm B	**apple**	2	0.909406

3 rows × 2 columns

In [20]:
```
def sim(grp):
    return DataFrame(np.random.normal(size=3), columns=['fruit per tree'])
fruit.groupby(['farm','fruit']).apply(sim)
```

Out[20]:

			fruit per tree
farm	**fruit**		
Farm A	**apple**	**0**	-1.118708
		1	0.166747
		2	-0.531929
	pear	**0**	-0.743631
		1	-0.655569
		2	-0.553312
Farm B	**apple**	**0**	-0.875545
		1	0.883906
		2	0.282449

9 rows × 1 columns

In [27]:
```
keys = ['farm','fruit','variety']
def sim(grp, size):
    return DataFrame(np.random.normal(grp['mean'],grp['sd'],size=size), columns=['fruit per tree'])

sim_fruit_per_tree = fruit.groupby(keys).apply(sim, 3)
sim_fruit_per_tree
```

Out[27]:

				fruit per tree
farm	**fruit**	**variety**		
Farm A	**apple**	**Granny Smith**	**0**	230.130175
			1	177.329619
			2	206.511362
		Jonathon	**0**	158.947631
			1	122.824318
			2	139.762772
	pear	**Nashi**	**0**	51.233758
			1	46.097449
			2	53.058483
Farm B	**apple**	**Granny Smith**	**0**	206.760033
			1	154.380444
			2	161.996342
		Jonathon	**0**	154.227499
			1	108.107354
			2	162.213587

15 rows × 1 columns

In [31]:
```
sim_fruit_per_tree = sim_fruit_per_tree.reset_index().rename(columns={'level_3':'iteration'})
sim_fruit_per_tree
```

Out[31]:

	farm	fruit	variety	iteration	fruit per tree
0	Farm A	apple	Granny Smith	0	230.130175
1	Farm A	apple	Granny Smith	1	177.329619
2	Farm A	apple	Granny Smith	2	206.511362
3	Farm A	apple	Jonathon	0	158.947631
4	Farm A	apple	Jonathon	1	122.824318
5	Farm A	apple	Jonathon	2	139.762772
6	Farm A	pear	Nashi	0	51.233758
7	Farm A	pear	Nashi	1	46.097449
8	Farm A	pear	Nashi	2	53.058483
9	Farm B	apple	Granny Smith	0	206.760033
10	Farm B	apple	Granny Smith	1	154.380444
11	Farm B	apple	Granny Smith	2	161.996342
12	Farm B	apple	Jonathon	0	154.227499
13	Farm B	apple	Jonathon	1	108.107354
14	Farm B	apple	Jonathon	2	162.213587

15 rows × 5 columns

In [29]:
```
N = 1000
sim2_fruit_per_tree = fruit.groupby(keys).apply(sim, N)
sim2_fruit_per_tree = sim2_fruit_per_tree.reset_index().rename(columns={'level_3':'iteration'})

sim2_fruit_per_tree.drop('iteration', axis=1).groupby(keys).describe().unstack()
```

Out[29]:

			fruit per tree							
			count	mean	std	min	25%	50%	75%	max
farm	**fruit**	**variety**								
Farm A	**apple**	**Granny Smith**	1000	198.976349	29.317363	111.245709	178.636120	198.595669	218.364087	291.925764
		Jonathon	1000	149.770180	22.068844	93.908548	134.214130	150.331061	164.927689	219.883789
	pear	**Nashi**	1000	50.024324	8.163186	26.928023	44.303949	49.907168	55.795602	74.941092
Farm B	**apple**	**Granny Smith**	1000	199.842617	29.973449	82.211135	179.849653	200.065176	219.643034	293.538759
		Jonathon	1000	149.589624	21.641614	78.585807	135.736765	150.431874	164.945984	213.109087

5 rows × 8 columns

Calculations

Merge the simulation with the number of trees to calculate the total fruit produced in each iteration

In [32]:
```
sim_fruit_per_tree.head()
```

Out[32]:

	farm	fruit	variety	iteration	fruit per tree
0	Farm A	apple	Granny Smith	0	230.130175
1	Farm A	apple	Granny Smith	1	177.329619
2	Farm A	apple	Granny Smith	2	206.511362
3	Farm A	apple	Jonathon	0	158.947631
4	Farm A	apple	Jonathon	1	122.824318

5 rows × 5 columns

In [33]:
```
sim_fruit = pd.merge(num_trees, sim_fruit_per_tree, how='left')
sim_fruit.head()
```

Out[33]:

	farm	fruit	variety	num trees	iteration	fruit per tree
0	Farm A	apple	Granny Smith	100	0	230.130175
1	Farm A	apple	Granny Smith	100	1	177.329619
2	Farm A	apple	Granny Smith	100	2	206.511362
3	Farm A	apple	Jonathon	200	0	158.947631
4	Farm A	apple	Jonathon	200	1	122.824318

5 rows × 6 columns

Add new columns with our calculations

In [34]:
```
sim_fruit["volume"] = sim_fruit["num trees"] * sim_fruit["fruit per tree"]
```

In [35]:
```
sim_fruit.head()
```

Out[35]:

	farm	fruit	variety	num trees	iteration	fruit per tree	volume
0	Farm A	apple	Granny Smith	100	0	230.130175	23013.017525
1	Farm A	apple	Granny Smith	100	1	177.329619	17732.961946
2	Farm A	apple	Granny Smith	100	2	206.511362	20651.136223
3	Farm A	apple	Jonathon	200	0	158.947631	31789.526143
4	Farm A	apple	Jonathon	200	1	122.824318	24564.863628

5 rows × 7 columns

Did you notice that `df[x]` refers to the *column* x, not the row?
You can use `df.ix[r,c]` to index using (row, column) order.

And so on to calculate profit...

In [36]:
```
# read prices, var and fixed costs, calculate profit
revenue_per_fruit = pd.read_csv('data/revenuePerFruit.csv', skipinitialspace=True)
fixed_cost = pd.read_csv('data/fixedCosts.csv', skipinitialspace=True)
cost_per_fruit = pd.read_csv('data/varCostsPerFruit.csv', skipinitialspace=True)
```

In [37]:
```
sim_var_cost = pd.merge(sim_fruit, cost_per_fruit)
sim_var_cost['var cost'] = sim_var_cost['volume'] * sim_var_cost['cost per fruit']
sim_var_cost.head()
```

Out[37]:

	farm	fruit	variety	num trees	iteration	fruit per tree	volume	cost per fruit	var cost
0	Farm A	apple	Granny Smith	100	0	230.130175	23013.017525	0.15	3451.952629
1	Farm A	apple	Granny Smith	100	1	177.329619	17732.961946	0.15	2659.944292
2	Farm A	apple	Granny Smith	100	2	206.511362	20651.136223	0.15	3097.670433
3	Farm A	apple	Jonathon	200	0	158.947631	31789.526143	0.15	4768.428921
4	Farm A	apple	Jonathon	200	1	122.824318	24564.863628	0.15	3684.729544

5 rows × 9 columns

In [38]:
```
sim_revenue = pd.merge(revenue_per_fruit, sim_fruit)
sim_revenue['revenue'] = sim_revenue['volume'] * sim_revenue['revenue per fruit']
sim_revenue.head()
```

Out[38]:

	fruit	variety	revenue per fruit	farm	num trees	iteration	fruit per tree	volume	revenue
0	apple	Granny Smith	0.9	Farm A	100	0	230.130175	23013.017525	20711.715772
1	apple	Granny Smith	0.9	Farm A	100	1	177.329619	17732.961946	15959.665751
2	apple	Granny Smith	0.9	Farm A	100	2	206.511362	20651.136223	18586.022600
3	apple	Granny Smith	0.9	Farm B	200	0	206.760033	41352.006590	37216.805931
4	apple	Granny Smith	0.9	Farm B	200	1	154.380444	30876.088826	27788.479944

5 rows × 9 columns

In [39]:
```
cashflows = pd.merge(sim_var_cost, sim_revenue, how='outer')
cashflows.head()
```

Out[39]:

	farm	fruit	variety	num trees	iteration	fruit per tree	volume	cost per fruit	var cost	revenue per fruit	revenue
0	Farm A	apple	Granny Smith	100	0	230.130175	23013.017525	0.15	3451.952629	0.90	20711.715772
1	Farm A	apple	Granny Smith	100	1	177.329619	17732.961946	0.15	2659.944292	0.90	15959.665751
2	Farm A	apple	Granny Smith	100	2	206.511362	20651.136223	0.15	3097.670433	0.90	18586.022600
3	Farm A	apple	Jonathon	200	0	158.947631	31789.526143	0.15	4768.428921	0.95	30200.049836
4	Farm A	apple	Jonathon	200	1	122.824318	24564.863628	0.15	3684.729544	0.95	23336.620447

5 rows × 11 columns

Aggregate

At this point we have all the cashflows at the `farm` × `fruit` × `variety` level.
But fixed costs are only defined at the `farm` level, so to go further, we need to aggregate.

In [40]:
```
fixed_cost
```

Out[40]:

	farm	fixed cost
0	Farm A	30000

1 rows × 2 columns

In [41]:
```
agg_cashflows = cashflows.groupby(['farm','iteration']).sum().reset_index()
agg_cashflows
```

Out[41]:

	farm	iteration	num trees	fruit per tree	volume	cost per fruit	var cost	revenue per fruit	revenue
0	Farm A	0	350	440.311564	57364.231568	0.50	8732.719130	2.88	53550.304145
1	Farm A	1	350	346.251386	44602.698000	0.50	6805.648321	2.88	41670.304797
2	Farm A	2	350	399.332617	51256.614764	0.50	7821.138422	2.88	47873.461146
3	Farm B	0	600	360.987532	103043.006100	0.36	18547.741098	1.85	95823.255465
4	Farm B	1	600	262.487798	74119.030415	0.36	13341.425475	1.85	68869.274453
5	Farm B	2	600	324.209929	97284.703335	0.36	17511.246600	1.85	90800.504746

6 rows × 9 columns

In [42]:
```
agg_cashflows = agg_cashflows.merge(fixed_cost, how='outer')
agg_cashflows
```

Out[42]:

	farm	iteration	num trees	fruit per tree	volume	cost per fruit	var cost	revenue per fruit	revenue	fixed cost
0	Farm A	0	350	440.311564	57364.231568	0.50	8732.719130	2.88	53550.304145	30000
1	Farm A	1	350	346.251386	44602.698000	0.50	6805.648321	2.88	41670.304797	30000
2	Farm A	2	350	399.332617	51256.614764	0.50	7821.138422	2.88	47873.461146	30000
3	Farm B	0	600	360.987532	103043.006100	0.36	18547.741098	1.85	95823.255465	NaN
4	Farm B	1	600	262.487798	74119.030415	0.36	13341.425475	1.85	68869.274453	NaN
5	Farm B	2	600	324.209929	97284.703335	0.36	17511.246600	1.85	90800.504746	NaN

6 rows × 10 columns

Easy to fix up those `NaN`s

In [43]:
```
agg_cashflows = agg_cashflows.fillna(0)
agg_cashflows
```

Out[43]:

	farm	iteration	num trees	fruit per tree	volume	cost per fruit	var cost	revenue per fruit	revenue	fixed cost
0	Farm A	0	350	440.311564	57364.231568	0.50	8732.719130	2.88	53550.304145	30000
1	Farm A	1	350	346.251386	44602.698000	0.50	6805.648321	2.88	41670.304797	30000
2	Farm A	2	350	399.332617	51256.614764	0.50	7821.138422	2.88	47873.461146	30000
3	Farm B	0	600	360.987532	103043.006100	0.36	18547.741098	1.85	95823.255465	0
4	Farm B	1	600	262.487798	74119.030415	0.36	13341.425475	1.85	68869.274453	0
5	Farm B	2	600	324.209929	97284.703335	0.36	17511.246600	1.85	90800.504746	0

6 rows × 10 columns

In [44]:
```
agg_cashflows['profit'] = agg_cashflows['revenue'] - agg_cashflows['fixed cost'] - agg_cashflows['var cost']
del agg_cashflows['fruit per tree'], agg_cashflows['cost per fruit'], agg_cashflows['revenue per fruit']
agg_cashflows
```

Out[44]:

	farm	iteration	num trees	volume	var cost	revenue	fixed cost	profit
0	Farm A	0	350	57364.231568	8732.719130	53550.304145	30000	14817.585015
1	Farm A	1	350	44602.698000	6805.648321	41670.304797	30000	4864.656475
2	Farm A	2	350	51256.614764	7821.138422	47873.461146	30000	10052.322724
3	Farm B	0	600	103043.006100	18547.741098	95823.255465	0	77275.514367
4	Farm B	1	600	74119.030415	13341.425475	68869.274453	0	55527.848979
5	Farm B	2	600	97284.703335	17511.246600	90800.504746	0	73289.258146

6 rows × 8 columns

In [45]:
```
iter_cashflows = agg_cashflows.groupby('iteration').sum()  # sum over farms
iter_cashflows
```

Out[45]:

	num trees	volume	var cost	revenue	fixed cost	profit
iteration						
0	950	160407.237668	27280.460228	149373.559611	30000	92093.099382
1	950	118721.728415	20147.073796	110539.579250	30000	60392.505454
2	950	148541.318098	25332.385022	138673.965893	30000	83341.580870

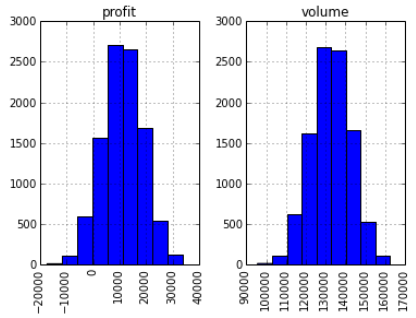
3 rows × 6 columns

All together

All the code in one place (except for a few small functions we defined)

In [46]:
```
N = 10000
keys = ['farm','fruit','variety']

# Read data
num_trees = pd.read_csv('data/numTrees.csv', skipinitialspace=True)
fruit_per_tree = pd.read_csv('data/fruitPerTree.csv', skipinitialspace=True)
revenue_per_fruit = pd.read_csv('data/revenuePerFruit.csv', skipinitialspace=True)
fixed_cost = pd.read_csv('data/fixedCosts.csv', skipinitialspace=True)
cost_per_fruit = pd.read_csv('data/varCostsPerFruit.csv', skipinitialspace=True)

# Simulate
def sim(grp, size):
    return DataFrame(np.random.normal(grp['mean'],grp['sd'],size=size), columns=['fruit per tree'])

sim_fruit_per_tree = fruit_per_tree.groupby(keys).apply(sim, N).reset_index().rename(columns={'level_3':'iteration'})

# Calculate var costs and revenues
sim_fruit = pd.merge(num_trees, sim_fruit_per_tree, how='left')
sim_fruit['volume'] = sim_fruit['fruit per tree'] * sim_fruit['num trees']
sim_var_cost = pd.merge(sim_fruit, cost_per_fruit)
sim_var_cost['var cost'] = sim_var_cost['volume'] * sim_var_cost['cost per fruit']
sim_revenue = pd.merge(revenue_per_fruit, sim_fruit)
sim_revenue['revenue'] = sim_revenue['volume'] * sim_revenue['revenue per fruit']
cashflows = pd.merge(sim_var_cost, sim_revenue, how='outer')
cashflows = cashflows.merge(fixed_cost, how='outer').fillna({'fixed cost':0})

# Aggregate and include fixed costs
agg_cashflows = cashflows.groupby(['farm','iteration']).sum().reset_index()
agg_cashflows = agg_cashflows.merge(fixed_cost, how='outer')
agg_cashflows['profit'] = agg_cashflows['revenue'] - agg_cashflows['fixed cost'] - agg_cashflows['var cost']
del agg_cashflows['fruit per tree'], agg_cashflows['cost per fruit'], agg_cashflows['revenue per fruit']

# Calculate profit
agg_cashflows['profit'] = agg_cashflows['revenue'] - agg_cashflows['fixed cost'] - agg_cashflows['var cost']
iter_cashflows = agg_cashflows.groupby('iteration').sum()
```

In [79]:
```
iter_cashflows[['profit','volume']].hist(xrot=90);
```



Based on these assumptions:

- the expected profit is 11,100
- the 80% confidence band is 1,100 to 21,000
- there is about an 8% chance of loss

In [48]:
```
iter_cashflows.describe(percentile_width=80).T
```

Out[48]:

	count	mean	std	min	10.0%	50%	90%	max
num trees	10000	950.000000	0.000000	950.000000	950.000000	950.000000	950.000000	950.000000
volume	10000	132531.002583	10077.223030	95614.986851	119407.202274	132508.424102	145229.107164	169735.090971
var cost	10000	22405.051342	1738.099191	16008.139927	20142.094211	22404.731080	24580.565159	28820.012684
revenue	10000	123503.212976	9425.981656	88576.492461	111277.416442	123449.475693	135410.669389	158317.758554
fixed cost	10000	90000.000000	0.000000	90000.000000	90000.000000	90000.000000	90000.000000	90000.000000
profit	10000	11098.161634	7694.926234	-17431.647466	1127.497613	11052.297244	20830.179123	39497.745870

6 rows × 8 columns

In [56]:
```
print "%3.1f%% chance of loss" % (100*(iter_cashflows['profit']<=0).sum()/(N+.0))  # uses Boolean indexing
```

```
7.8% chance of loss
```

In [ ]:
```

```

We have used these key pandas features:

- `read_csv`
- `merge`
- `groupby` & `apply`
- `fillna`
- `hist`
- Broadcasting
- Boolean indexing

And:

- `drop`
- `reset_index`
- `rename`
- `describe`
- `unstack`

Montepylib

In [57]:
```
import os
os.sys.path.append(os.path.dirname(os.path.abspath('..')))
from examples import example1
```

In [58]:
```
data = example1.read_data("../../examples/data/")
```

In [59]:
```
data['fruit_per_tree']  # similar to before, but by year
```

Out[59]:

	farm	fruit	variety	year	mean	sd
0	Farm A	apple	Granny Smith	2014	200	30.00
1	Farm A	apple	Jonathon	2014	150	22.50
2	Farm A	pear	Nashi	2014	50	7.50
3	Farm B	apple	Granny Smith	2014	160	24.00
4	Farm B	apple	Jonathon	2014	120	18.00
5	Farm A	apple	Granny Smith	2015	220	33.00
6	Farm A	apple	Jonathon	2015	175	26.25
7	Farm A	pear	Nashi	2015	55	8.25
8	Farm B	apple	Granny Smith	2015	180	27.00
9	Farm B	apple	Jonathon	2015	150	22.50
10	Farm A	apple	Granny Smith	2016	240	36.00
11	Farm A	apple	Jonathon	2016	200	30.00
12	Farm A	pear	Nashi	2016	60	9.00
13	Farm B	apple	Granny Smith	2016	200	30.00
14	Farm B	apple	Jonathon	2016	180	27.00
15	Farm A	apple	Granny Smith	2017	250	37.50
16	Farm A	apple	Jonathon	2017	225	33.75
17	Farm A	pear	Nashi	2017	60	9.00
18	Farm B	apple	Granny Smith	2017	220	33.00
19	Farm B	apple	Jonathon	2017	200	30.00

20 rows × 6 columns

In [60]:
```
# the base case (no simulation)
example1.simulate(data, N=0)
```

Out[60]:

		num trees	fruit per tree	volume	cost per fruit	var cost	revenue per fruit	revenue	fixed cost	profit
iteration	**year**									
base case	**2014**	950	680	132500	0.86	22400	4.70	123400	90000	11000
	2015	950	780	155750	0.86	26380	4.90	151430	90000	35050
	2016	950	880	179000	0.86	30360	5.08	180680	90000	60320
	2017	950	955	197000	0.86	33420	5.28	206800	90000	83380

4 rows × 9 columns

In [61]:
```
# one-factor sensitivities
sensitivities = example1.simulate(data, N=-1)
sensitivities
```

Out[61]:

		cost per fruit	fixed cost	fruit per tree	num trees	revenue	revenue per fruit	var cost	volume	profit
iteration	**year**									
base case	**2014**	0.86	90000	680	950	123400	4.700	22400	132500	11000
	2015	0.86	90000	780	950	151430	4.900	26380	155750	35050
	2016	0.86	90000	880	950	180680	5.080	30360	179000	60320
	2017	0.86	90000	955	950	206800	5.280	33420	197000	83380
high fixed cost	**2014**	0.86	90000	680	950	123400	4.700	22400	132500	11000
	2015	0.86	90000	780	950	151430	4.900	26380	155750	35050
	2016	0.86	90000	880	950	180680	5.080	30360	179000	60320
	2017	0.86	90000	955	950	206800	5.280	33420	197000	83380
high revenue per fruit	**2014**	0.86	90000	680	950	148080	5.640	22400	132500	35680
	2015	0.86	90000	780	950	181716	5.880	26380	155750	65336
	2016	0.86	90000	880	950	216816	6.096	30360	179000	96456
	2017	0.86	90000	955	950	248160	6.336	33420	197000	124740
low fixed cost	**2014**	0.86	90000	680	950	123400	4.700	22400	132500	11000
	2015	0.86	90000	780	950	151430	4.900	26380	155750	35050
	2016	0.86	90000	880	950	180680	5.080	30360	179000	60320
	2017	0.86	90000	955	950	206800	5.280	33420	197000	83380
low revenue per fruit	**2014**	0.86	90000	680	950	98720	3.760	22400	132500	-13680
	2015	0.86	90000	780	950	121144	3.920	26380	155750	4764
	2016	0.86	90000	880	950	144544	4.064	30360	179000	24184
	2017	0.86	90000	955	950	165440	4.224	33420	197000	42020

20 rows × 9 columns

In [62]:
```
# full simulation with NPV calculation
N = 1000
total_per_iteration = example1.simulate(data, N, discount_rate=1.09, base_year=2014)
```

In [63]:
```
# example1.sim_output(total_per_iteration, N=1000)
# the example output is:
total_per_iteration.xs('2017', level='year').describe(percentile_width=80).T
```

Out[63]:

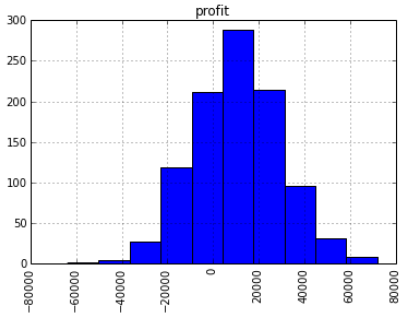
	count	mean	std	min	10.0%	50%	90%	max
num trees	1000	950.000000	0.000000	950.000000	950.000000	950.000000	950.000000	950.000000
fruit per tree	1000	956.971171	70.561931	762.397112	862.889296	958.357739	1043.667005	1253.695033
volume	1000	197156.863761	16665.839963	146779.042820	175219.295045	197257.724424	218751.500684	268160.548834
cost per fruit	1000	0.860000	0.000000	0.860000	0.860000	0.860000	0.860000	0.860000
var cost	1000	33445.297851	2889.865534	24669.381359	29644.206583	33483.367807	37248.647198	45775.678659
revenue per fruit	1000	5.270118	0.598442	3.334018	4.476001	5.279304	6.010670	7.336881
revenue	1000	205977.316765	33825.977248	96935.914581	163613.290877	203543.695460	248732.687212	326867.809221
fixed cost	1000	90000.000000	0.000000	90000.000000	90000.000000	90000.000000	90000.000000	90000.000000
profit	1000	82532.018913	32461.113966	-25193.699049	42029.555969	80669.712054	124297.498582	198214.032854
discount factor	1000	0.772183	0.000000	0.772183	0.772183	0.772183	0.772183	0.772183
discounted profit	1000	63729.861581	25065.935949	-19454.158207	32454.528793	62291.818989	95980.475018	153057.601686

11 rows × 8 columns

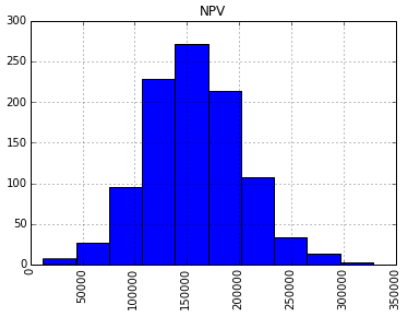
In [67]:
```
print "%3.1f%% chance of loss in 2014" % (100*(total_per_iteration.xs('2014', level='year')['profit']<=0).sum()/(N+.0))
```

```
28.6% chance of loss in 2014
```

In [77]:
```
# histogram of profit in 2014
total_per_iteration.xs('2014', level='year')[['profit']].hist(xrot=90);
```



In [78]:
```
#histogram of NPV
example1.calc_NPVs(total_per_iteration).hist(xrot=90);
```



And back to some final slides...

.
.
.
.
.
.
.
.
.
.

Styling

In [37]:
```
from IPython.core.display import HTML
def css_styling():
    styles = open("./css/notebook.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[37]:

In [37]: