

# Cambridge Communications Assessment Model

## Prototype Mobile/Wireless Model

**Dr Edward J. Oughton, Judge Business School, University of Cambridge**

**Email address: [e.oughton@jbs.cam.ac.uk](mailto:e.oughton@jbs.cam.ac.uk)**

Mobile communications are essential for everyday activity, and underpin both the economy and our society. This workbook provides a prototype mobile/wireless model, the aim of which is to develop long-term national infrastructure strategies for the UK.

Currently, synthetic data is being used to illustrate the structure and functionality of the model, with the aim of completing development by the end of 2017. This forms one part of the Cambridge Communications Assessment Model. This prototype model can be used to generate (simplistic and dummy) results for:

- *capacity-demand by area*
- *user throughput by area*
- *aggregate demand*
- *capacity margin*
- *area demand distribution*
- *capacity margin distribution*
- *capacity margin annual change*
- *capacity margin cumulative sum*
- *cumulative cost of meeting new demand*
- *energy demand*

## Packages and global parameters

First, let's import the packages required, which will focus on 'Numpy', 'Pandas' and 'Matplotlib'.

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

We also want to set the seed for the pseudo random number generator, so we can maintain the same (pseudo)randomly generated dataset. Here, '42' is used as the seed number as this relates to a famous book.

```
In [2]: np.random.seed(seed=42)
```

We also want to set some key global parameters that will dictate the length of the dummy data we intend to generate, and the number of forecast steps we wish to use.

```
In [3]: #set length of dummy data set  
length = 5  
  
#set number of years to forecast  
k = 8
```

## Calculating demand

Next we want to set up the demand variables. These need to represent actual demand for each individual local area unit, which for now we use synthetic data (until the model structure has been finalised).

First we include an area 'ID', and then we generate statistics for the total 'population' and geographical 'area'. The 'user\_demand' needs to be set (the total number of Gigabytes of data downloaded per month, per user), along with the user 'penetration' rate, which is assumed here to be static at 80% of the local population.

Finally, we use an Over Booking Factor('OBF') of 50, indicating that one in every fifty users is using the digital network at the same time.

```
In [4]: #define dummy input data and dummy variables for demand  
ID = np.arange(length)  
population = np.random.randint(low=4000, high=10000, size=length)  
area = np.random.randint(low=1, high=50, size=length)  
user_demand = [4]*length  
penetration = 0.8  
OBF = 50
```

Once these metrics have been generated, we use a function to calculate the key metrics of interest for our demand calculations, which include total 'users', user 'density' per km<sup>2</sup>, total 'user\_throughput', the area capacity needed ('capacity\_required\_km2'), and finally the total 'area\_demand'. All metrics are concatenated into a Pandas dataframe.

```

In [5]: #define demand function
def calc_demand (population, area, user_demand, penetration):
    users = population*penetration
    density = np.round(users / area, 2) #km2
    user_throughput = user_demand #temporary calculation!!
    capacity_required_km2 = user_throughput * density
    area_demand = np.round(capacity_required_km2 / OBF, 2)
    return pd.DataFrame({'ID':ID,
                        'population':population,
                        'area': area,
                        #'users': users,
                        #'density': density,
                        'user_throughput': user_throughput,
                        'area_demand': area_demand})

#when you call the function, store the output as an object
area_demand = calc_demand(population, area, user_demand, penetration)

#print head of df
area_demand.head(n=4)

```

```

Out[5]:

```

	ID	area	area_demand	population	user_throughput
0	0	21	14.81	4860	4
1	1	39	15.41	9390	4
2	2	19	31.08	9226	4
3	3	23	25.58	9191	4

As this is a prototype model, we then want to generate some temporal data using some modest time-series growth trends.

This is specified for demand growth, population growth and user\_throughput growth. Importantly, future versions of this model need to improve on this by generating time-series for the underlying variables, with the capacity calculation being undertaken at each time step.

Demand growth of 20% is plausible for digital, along with a 15% year-on-year increase in user\_throughput. 10% population growth is very high, but we keep it like this for now for the sake of having a working prototype example.

```

In [6]: #temporal demand growth
a = 1.20 #Demand Growth
b = 1.10 #Population Growth
c = 1.15 #User_throughput

```

Now we have specified our annual growth rates, we now apply them year-on-year for the number of time steps specified, for each area.

Once complete, we sort the values by year and area ID.

```

In [7]: area_demand =
        (area_demand[['ID', 'area']].merge(pd.concat([(area_demand[['area_demand', 'p
lation', 'user_throughput']]).mul([pow(a,i), pow(b,i), pow(c,i)])))).assign(ye
ar=i+1) for i in range(k)),
        left_index=True, right_index=True)
        .sort_values(by='year'))

area_demand.sort_values(['year', 'ID'], ascending=[True, True], inplace=Tr
ue)

area_demand.head(n=10)

```

Out[7]:

	ID	area	area_demand	population	user_throughput	year
0	0	21	14.810	4860.0	4.0	1
1	1	39	15.410	9390.0	4.0	1
2	2	19	31.080	9226.0	4.0	1
3	3	23	25.580	9191.0	4.0	1
4	4	11	45.220	7772.0	4.0	1
0	0	21	17.772	5346.0	4.6	2
1	1	39	18.492	10329.0	4.6	2
2	2	19	37.296	10148.6	4.6	2
3	3	23	30.696	10110.1	4.6	2
4	4	11	54.264	8549.2	4.6	2

We can now verify the shape of the pandas dataframe using:

```
In [8]: area_demand.shape
```

```
Out[8]: (40, 6)
```

## Caculating supply

The key factors affecting the supply of mobile/wireless capacity are spectrum availability, sites density and spectral efficiency.

Here we generate some dummy data, representing 20 MHz spectrum bandwidth, a site density between 1-10 per km<sup>2</sup>, and an LTE-like spectral efficiency of 3 bits per hertz.

```
In [9]: #define dummy input data and dummy variables for supply
spectrum_bandwidth = [20]*length
sites = np.random.randint(low=1, high=10, size=length)
efficiency = [3]*length
```

We then need to generate the supply metrics desired to calculate actual capacity per km<sup>2</sup>. We assume each site is a three-sectored macro basestation. Total capacity is calculated by multiplying bandwidth by the number of sites, cells and the spectral efficiency.

As this coverage-capacity matter spatially, we convert it to the 'area\_supply' in km<sup>2</sup>.

```
In [10]: def calc_supply (spectrum_bandwidth, sites, efficiency):
          ID = np.arange(length)
          cells = sites * 3
          total_capacity = spectrum_bandwidth * sites * cells * efficiency/1000
          area_supply = np.round(total_capacity / area, 0)
          return pd.DataFrame({'ID':ID,
                               'sites':sites,
                               'cells': cells,
                               'area_supply': area_supply,
                               'new_sites': '' })

          area_supply = calc_supply(spectrum_bandwidth, sites, efficiency)

          area_supply.head(n=4)
```

```
Out[10]:
```

	ID	area_supply	cells	new_sites	sites
0	0	1.0	24		8
1	1	0.0	15		5
2	2	0.0	12		4
3	3	1.0	24		8

In this example we assume there is no future supply growth, setting the temporal trends for each supply metric to 1.

```
In [11]: #temporal supply growth
a = 1.00 #supply growth
b = 1.00 #cells
c = 1.00 #new sites growth
#d = 1.00 #sites growth
```

Much like with the demand, we then take this one year snapshot and turn it into a multi-year time series for each area.

```
In [12]: area_supply = (area_supply[['ID','new_sites']].merge(pd.concat([(area_supply[['area_supply','cells','sites']].mul([pow(a,i),pow(b,i), pow(c,i)]))
                                                                    .assign(year=i+1) for i in range(k)]),
                                                                    left_index=True, right_index=True)
                                                                    .sort_values(by='year'))

area_supply.sort_values(['year', 'ID'], ascending=[True, True], inplace=True)

area_supply.head(n=10)
```

```
Out[12]:
```

	ID	new_sites	area_supply	cells	sites	year
0	0		1.0	24.0	8.0	1
1	1		0.0	15.0	5.0	1
2	2		0.0	12.0	4.0	1
3	3		1.0	24.0	8.0	1
4	4		1.0	24.0	8.0	1
0	0		1.0	24.0	8.0	2
1	1		0.0	15.0	5.0	2
2	2		0.0	12.0	4.0	2
3	3		1.0	24.0	8.0	2
4	4		1.0	24.0	8.0	2

Again, we can verify the shape of the data using:

```
In [13]: area_supply.shape
```

```
Out[13]: (40, 6)
```

## Calculate capacity margin

We then want to calculate the capacity margin by area, by subtracting demand from supply. We specify the following function and then use it to generate a vector of capacity margin values.

```
In [14]: def calc_margin (ID, demand, supply, year):
margin = supply - demand
year = year
return pd.DataFrame({'margin':margin})

margin = calc_margin(area_demand.ID, area_demand.area_demand,
area_supply.area_supply, area_supply.year)

margin.head(n=4)
```

```
Out[14]:
```

	margin
0	-13.81
1	-15.41
2	-31.08
3	-24.58

To avoid multiple columns with the same values, we drop ID and year from 'area\_supply' and then concatenate with 'area\_demand' and 'margin' to generate our total dataset.

```
In [15]: #round
area_supply = area_supply.drop('ID', 1)
area_supply = area_supply.drop('year', 1)

#concat all data
all_data = pd.concat([area_demand, area_supply, margin], axis=1)

all_data.head(n=5)
```

```
Out[15]:
```

	ID	area	area_demand	population	user_throughput	year	new_sites	area_supply
0	0	21	14.81	4860.0	4.0	1		1.0
1	1	39	15.41	9390.0	4.0	1		0.0
2	2	19	31.08	9226.0	4.0	1		0.0
3	3	23	25.58	9191.0	4.0	1		1.0
4	4	11	45.22	7772.0	4.0	1		1.0

Next, we introduce a simple lookup table called 'lookup\_df' that relates Inter-Site Distance (hence, network density), to supply capacity.

For now we focus on site density, but spectrum integration is an important factor that needs to be included in the next iteration of the model.

```
In [16]: #put a name on the margin column as it was previously not named
#all_data.rename(columns={0:'margin'}, inplace=True)

lookup_df = pd.DataFrame({'spectrum_lookup' : (2,4,6,8,10,12,14,16),
                           'sites_lookup' : (5,10,15,20,25,30,35,40),
                           'supply_lookup' :
                           (50,100,150,200,250,300,350,400)})

lookup_df.head(n=5)
```

Out[16]:

	sites_lookup	spectrum_lookup	supply_lookup
0	5	2	50
1	10	4	100
2	15	6	150
3	20	8	200
4	25	10	250

A function is then used which takes any rows that have a capacity margin deficit, and takes the number of sites from the 'lookup\_df' required to meet the specific demand of that area.

The 'final\_number' of new sites is calculated (based on subtracting the lookup sites value from the number of existing sites) and placed in the 'new\_sites' column of 'all\_data'



```

In [17]: def lookup_new_sites(row):
            if row['margin'] < 0:
                # get the minimum number of sites that cover the current supply de
                ficit
                number_new = lookup_df.loc[lookup_df['supply_lookup'] >=
                abs(row['area_demand']), 'sites_lookup'].values[0]
                final_number = number_new - row['sites']
                if final_number < 0:
                    return 0
                else:
                    return final_number
            else:
                return 0

all_data['new_sites'] = all_data.apply(lookup_new_sites, axis=1)

# all_data['new_sites'] =int(yahoostock.get_price('RIL.BO'));

all_data.head(n=10)

```

Out[17]:

	ID	area	area_demand	population	user_throughput	year	new_sites	area_supply
0	0	21	14.810	4860.0	4.0	1	0.0	1.0
1	1	39	15.410	9390.0	4.0	1	0.0	0.0
2	2	19	31.080	9226.0	4.0	1	1.0	0.0
3	3	23	25.580	9191.0	4.0	1	0.0	1.0
4	4	11	45.220	7772.0	4.0	1	0.0	1.0
0	0	21	17.772	5346.0	4.6	2	0.0	1.0
1	1	39	18.492	10329.0	4.6	2	0.0	0.0
2	2	19	37.296	10148.6	4.6	2	1.0	0.0
3	3	23	30.696	10110.1	4.6	2	0.0	1.0
4	4	11	54.264	8549.2	4.6	2	2.0	1.0

Once complete, we also want to be able to calculate the year-on-year difference, which is placed in the 'annual\_change' column of the 'output' dataframe as follows.

```
In [18]: all_data = all_data.reset_index()

output = all_data.assign(annual_change=all_data.groupby("ID")
['new_sites'].apply(lambda x:x.diff().fillna(x)))

output.head(n=10)
```

Out[18]:

	index	ID	area	area_demand	population	user_throughput	year	new_sites	area_
0	0	0	21	14.810	4860.0	4.0	1	0.0	1.0
1	1	1	39	15.410	9390.0	4.0	1	0.0	0.0
2	2	2	19	31.080	9226.0	4.0	1	1.0	0.0
3	3	3	23	25.580	9191.0	4.0	1	0.0	1.0
4	4	4	11	45.220	7772.0	4.0	1	0.0	1.0
5	0	0	21	17.772	5346.0	4.6	2	0.0	1.0
6	1	1	39	18.492	10329.0	4.6	2	0.0	0.0
7	2	2	19	37.296	10148.6	4.6	2	1.0	0.0
8	3	3	23	30.696	10110.1	4.6	2	0.0	1.0
9	4	4	11	54.264	8549.2	4.6	2	2.0	1.0

Having calculated the number of new cells required, we can then work out both the cost, and the energy demand for these new assets.

We introduce a dataframe 'cells' which (currently crudely) represents the Total Cost of Ownership (TCO) for new assets. We do this for both capex and opex.

## Capex lookup

The 'capex' lookup contains dummy data now that can be populated with real data once the structure of the model has been finalised.

```

In [19]: cells = {'4G LTE-A MaBS' : pd.Series([30, 30, 30], index=['RAN', 'transmission', 'site']),
                  '4G LTE-A MiBS' : pd.Series([15,30,10], index=['RAN', 'transmission', 'site']),
                  '5G mmW MiBS' : pd.Series([8,30,8], index=['RAN', 'transmission', 'site']),
                  '5G mmW PBS' : pd.Series([6,10,2.5], index=['RAN', 'transmission', 'site']),
                  'Wi-Fi IEEE 802.11ac AP' : pd.Series([2.5,5,1], index=['RAN', 'transmission', 'site']),
                  'Wi-Fi IEEE 802.11ad AP' : pd.Series([2.5,5,1], index=['RAN', 'transmission', 'site'])}

capex = pd.DataFrame(cells)

capex = capex.T

capex=capex.rename(columns = {'index':'type'})

capex['TCO'] = capex.sum(axis=1)

capex.head(n=10)

```

Out[19]:

	RAN	transmission	site	TCO
<b>4G LTE-A MaBS</b>	30.0	30.0	30.0	90.0
<b>4G LTE-A MiBS</b>	15.0	30.0	10.0	55.0
<b>5G mmW MiBS</b>	8.0	30.0	8.0	46.0
<b>5G mmW PBS</b>	6.0	10.0	2.5	18.5
<b>Wi-Fi IEEE 802.11ac AP</b>	2.5	5.0	1.0	8.5
<b>Wi-Fi IEEE 802.11ad AP</b>	2.5	5.0	1.0	8.5

## Opex lookup

The 'opex' lookup contains dummy data now that can be populated with real data once the structure of the model has been finalised.

```

In [20]: cells = {'4G LTE-A MaBS' : pd.Series([30, 30, 30], index=['transmission',
'site', 'O&M power']),
                  '4G LTE-A MiBS' : pd.Series([15,30,10], index=['transmission',
'site', 'O&M power']),
                  '5G mmW MiBS' : pd.Series([8,30,8], index=['transmission', 'site', 'O
&M power']),
                  '5G mmW PBS' : pd.Series([6,10,2.5], index=['transmission', 'site',
'O&M power']),
                  'Wi-Fi IEEE 802.11ac AP' : pd.Series([2.5,5,1], index=
['transmission', 'site', 'O&M power']),
                  'Wi-Fi IEEE 802.11ad AP' : pd.Series([2.5,5,1], index=
['transmission', 'site', 'O&M power'])}

opex = pd.DataFrame(cells)

opex = opex.T

opex=opex.rename(columns = {'index':'type'})

opex['TCO'] = opex.sum(axis=1)

opex.head(n=5)

```

```

Out[20]:

```

	transmission	site	O&M power	TCO
<b>4G LTE-A MaBS</b>	30.0	30.0	30.0	90.0
<b>4G LTE-A MiBS</b>	15.0	30.0	10.0	55.0
<b>5G mmW MiBS</b>	8.0	30.0	8.0	46.0
<b>5G mmW PBS</b>	6.0	10.0	2.5	18.5
<b>Wi-Fi IEEE 802.11ac AP</b>	2.5	5.0	1.0	8.5

## Energy demand lookup

The 'energy\_lookup' contains dummy data now that can be populated with real data once the structure of the model has been finalised.

```

In [21]: cells = {'4G LTE-A MaBS' : pd.Series([30], index=['energy_demand']),
                  '4G LTE-A MiBS' : pd.Series([15], index=['energy_demand']),
                  '5G mmW MiBS' : pd.Series([8], index=['energy_demand']),
                  '5G mmW PBS' : pd.Series([6], index=['energy_demand']),
                  'Wi-Fi IEEE 802.11ac AP' : pd.Series([2.5], index=['energy_demand']),
                  'Wi-Fi IEEE 802.11ad AP' : pd.Series([2.5], index=['energy_demand'])}

energy_lookup = pd.DataFrame(cells)

energy_lookup = energy_lookup.T

#opex.reset_index(level=0, inplace=True)

energy_lookup = energy_lookup.rename(columns = {'index':'type'})

energy_lookup.head(n=5)

```

Out[21]:

	energy_demand
<b>4G LTE-A MaBS</b>	30.0
<b>4G LTE-A MiBS</b>	15.0
<b>5G mmW MiBS</b>	8.0
<b>5G mmW PBS</b>	6.0
<b>Wi-Fi IEEE 802.11ac AP</b>	2.5

## Calculate TCO

Now we calculate the TCO using 'lookup\_df', currently just focusing on LTE-A Macro BS.

```
In [22]: def calculate_TCO(new_sites):
        cap_cost = capex.loc[["4G LTE-A MaBS","type"], "TCO"].values[0]
        op_cost = opex.loc[["4G LTE-A MaBS","type"], "TCO"].values[0]
        total_cap_cost = new_sites * cap_cost
        total_op_cost = new_sites * op_cost
        TCO = total_cap_cost + total_op_cost
        return TCO

total_cost = calculate_TCO(output.new_sites)

total_cost= pd.DataFrame(total_cost)

total_cost = total_cost.rename(columns = {'new_sites':'TCO'})

total_cost.head(n=5)
```

Out[22]:

	TCO
0	0.0
1	0.0
2	180.0
3	0.0
4	0.0

And, now we can concatenate the results with the existing 'output' df.

```
In [23]: output = pd.concat([output, total_cost], axis=1)

output.head(n=5)
```

Out[23]:

	index	ID	area	area_demand	population	user_throughput	year	new_sites	area_
0	0	0	21	14.81	4860.0	4.0	1	0.0	1.0
1	1	1	39	15.41	9390.0	4.0	1	0.0	0.0
2	2	2	19	31.08	9226.0	4.0	1	1.0	0.0
3	3	3	23	25.58	9191.0	4.0	1	0.0	1.0
4	4	4	11	45.22	7772.0	4.0	1	0.0	1.0

## Calculate energy demand

Now we calculate the energy\_demand of new BS using 'energy\_lookup', still just focusing on LTE-A Macro BS.

```
In [24]: def calculate_energy_demand(sites, new_sites):
            demand_per_asset = energy_lookup.loc[["4G LTE-A MaBS", "type"], "energy_demand"].values[0]
            total_energy_demand = (sites + new_sites) * demand_per_asset
            #does not include decommissioning currently, but this is very important
            return pd.DataFrame({'total_energy_demand':total_energy_demand})

energy_demand = calculate_energy_demand(output.sites, output.new_sites)
energy_demand = pd.DataFrame(energy_demand)
energy_demand = energy_demand.rename(columns = {'0':'energy_demand'})
energy_demand.head(n=5)
```

```
Out[24]:
```

	total_energy_demand
0	240.0
1	150.0
2	150.0
3	240.0
4	240.0

And, now we can concatenate the results with the existing 'output' df.

```
In [25]: output = pd.concat([output, energy_demand], axis=1)
output.head(n=5)
```

```
Out[25]:
```

	index	ID	area	area_demand	population	user_throughput	year	new_sites	area_
0	0	0	21	14.81	4860.0	4.0	1	0.0	1.0
1	1	1	39	15.41	9390.0	4.0	1	0.0	0.0
2	2	2	19	31.08	9226.0	4.0	1	1.0	0.0
3	3	3	23	25.58	9191.0	4.0	1	0.0	1.0
4	4	4	11	45.22	7772.0	4.0	1	0.0	1.0

## Generate output graphics

Now we can generate output graphics based on the underlying data.

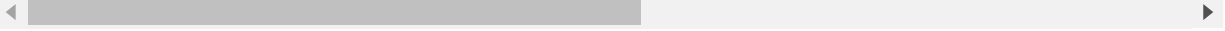
First we need to generate some additional aggregate statistics.

```
In [26]: #sum by year
cap_margin = output.groupby(['year'], as_index=False).sum()

cap_margin.head(n=5)
```

Out[26]:

	year	index	ID	area	area_demand	population	user_throughput	new_sites	area_
0	1	10	10	113	132.10000	40439.0000	20.000000	1.0	3.0
1	2	10	10	113	158.52000	44482.9000	23.000000	3.0	3.0
2	3	10	10	113	190.22400	48931.1900	26.450000	3.0	3.0
3	4	10	10	113	228.26880	53824.3090	30.417500	8.0	3.0
4	5	10	10	113	273.92256	59206.7399	34.980125	10.0	3.0



## Demand by area

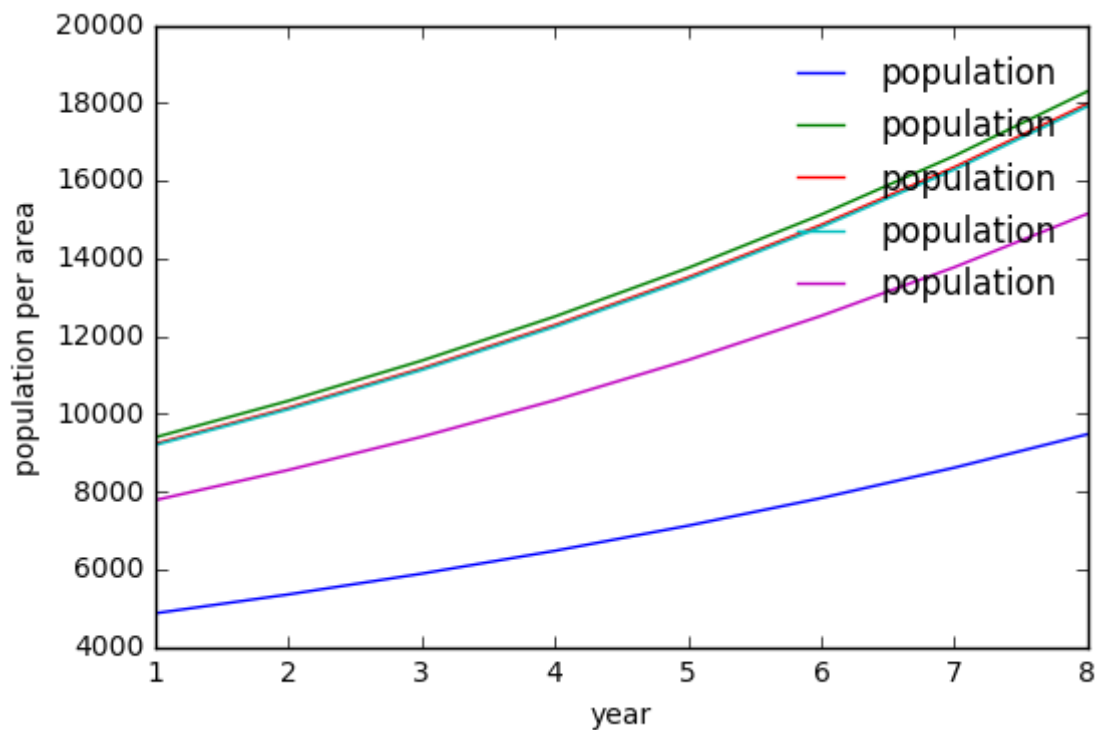


```

In [27]: #area demand for a region, LA or MSOA
plt.plot(output.year[output.area == 21],output.population [output.area ==
21])
plt.plot(output.year[output.area == 39],output.population [output.area ==
39])
plt.plot(output.year[output.area == 19],output.population [output.area ==
19])
plt.plot(output.year[output.area == 23],output.population [output.area ==
23])
plt.plot(output.year[output.area == 11],output.population [output.area ==
11])
plt.xlabel('year')
plt.ylabel('population per area')
plt.legend(framealpha=0, frameon=False)

```

Out[27]: <matplotlib.legend.Legend at 0x93e1ef0>



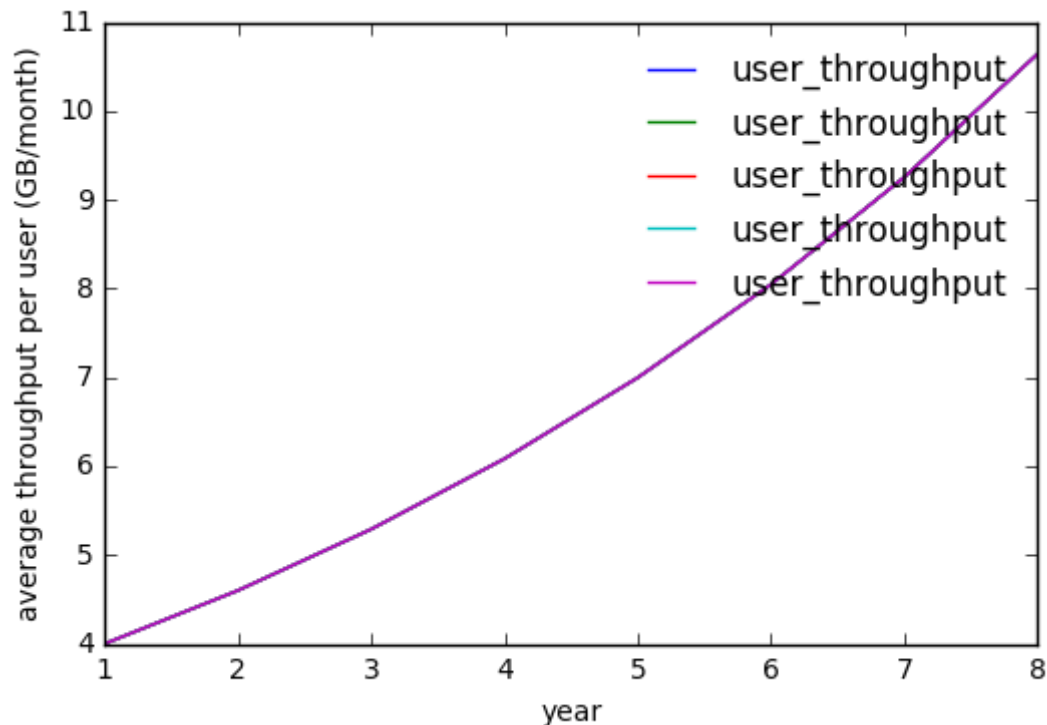
## Average throughput per user

```

In [28]: #user_throughput
#currently there is only one user type, but demand can be scaled based on
#demographic group
plt.plot(output.year[output.area == 21],output.user_throughput [output.area
a == 21])
plt.plot(output.year[output.area == 39],output.user_throughput [output.area
a == 39])
plt.plot(output.year[output.area == 19],output.user_throughput [output.area
a == 19])
plt.plot(output.year[output.area == 23],output.user_throughput [output.area
a == 23])
plt.plot(output.year[output.area == 11],output.user_throughput [output.area
a == 11])
plt.xlabel('year')
plt.ylabel('average throughput per user (GB/month)')
plt.legend(framealpha=0, frameon=False)

```

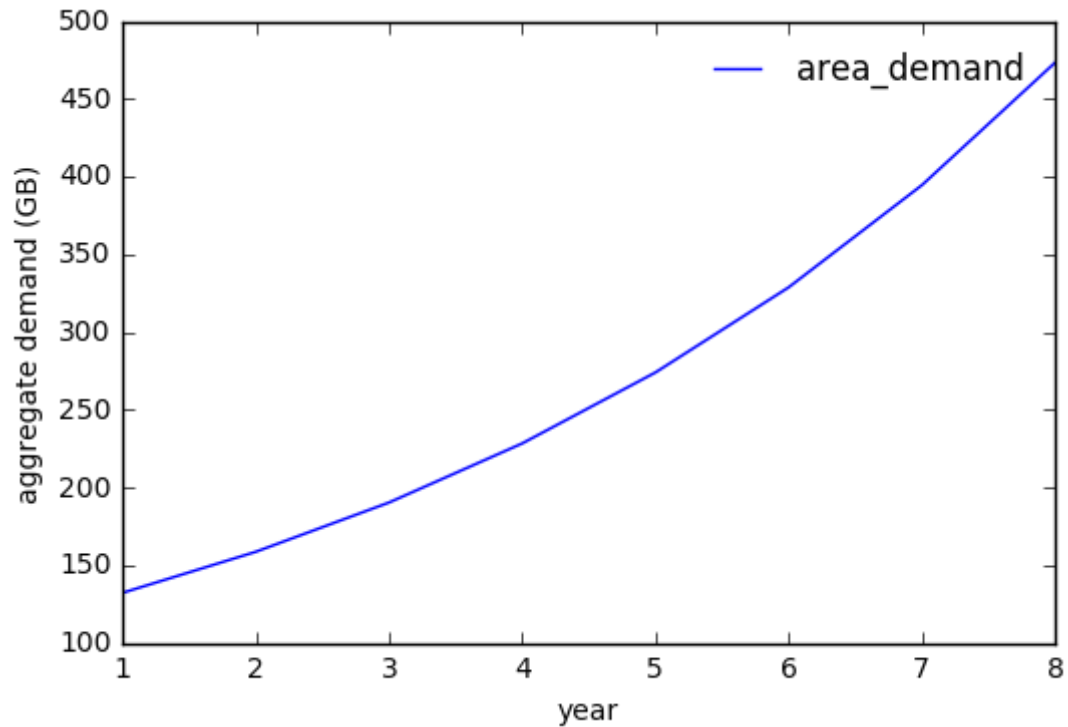
Out[28]: <matplotlib.legend.Legend at 0x4847e80>



## Aggregate demand

```
In [29]: #aggregate demand
plt.plot(cap_margin.year,cap_margin.area_demand)
plt.xlabel('year')
plt.ylabel('aggregate demand (GB)')
plt.legend(framealpha=0, frameon=False)
```

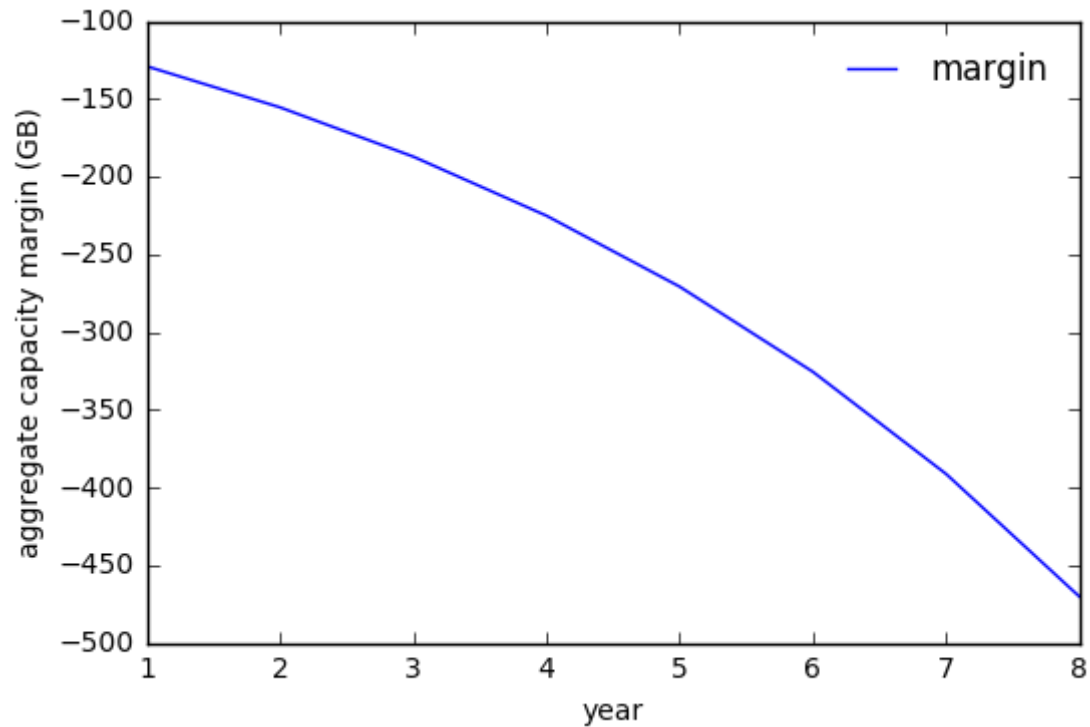
Out[29]: <matplotlib.legend.Legend at 0x91d41d0>



## Capacity margin

```
In [30]: #capacity margin (resembling a no-build scenario)
plt.plot(cap_margin.year, cap_margin.margin)
plt.xlabel('year')
plt.ylabel('aggregate capacity margin (GB)')
plt.legend(framealpha=0, frameon=False)
```

Out[30]: <matplotlib.legend.Legend at 0x9665dd8>

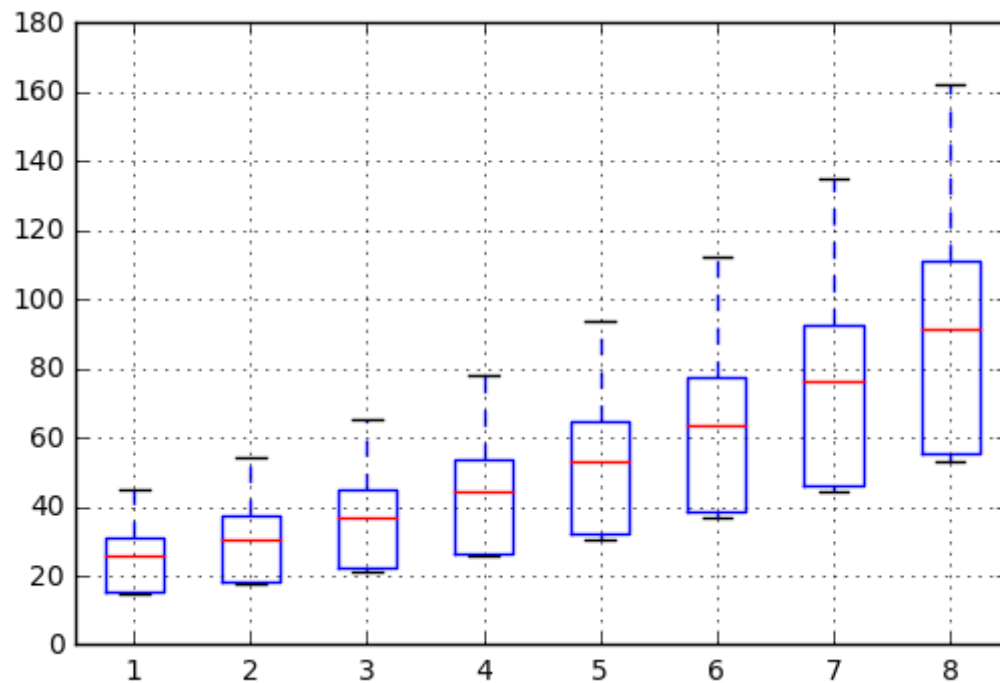


## Area demand distribution

```
In [31]: #area_demand distribution
coverage = output.pivot(index='ID', columns='year', values='area_demand')

plt.figure()

dp = coverage.boxplot()
```



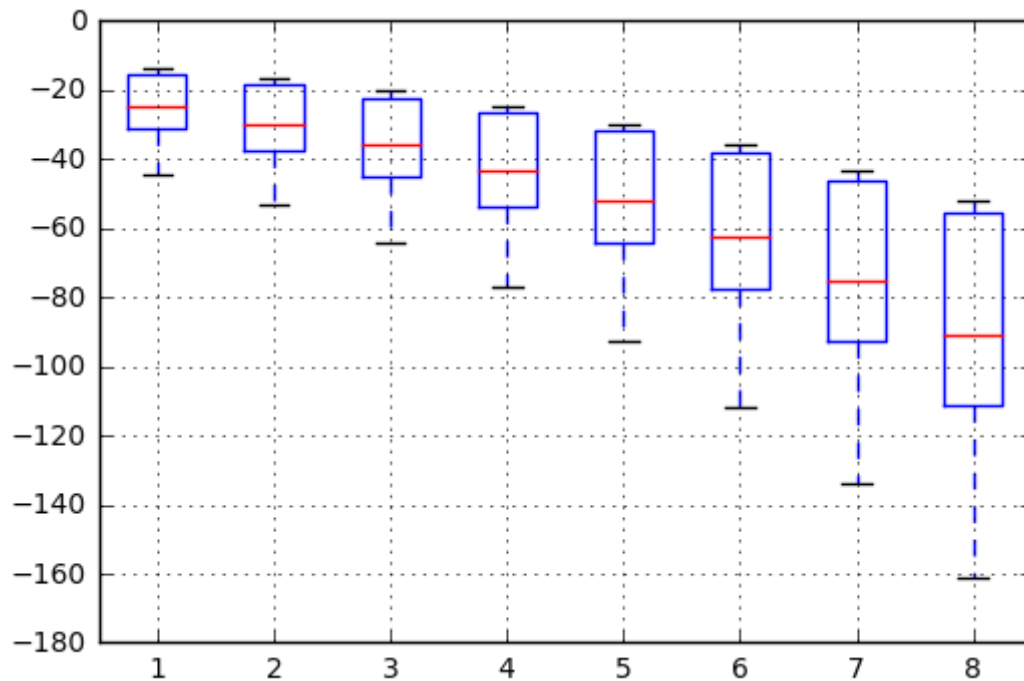
```
In [32]: ## Area capacity margin distribution
```

```
In [33]: #area_capacity_margin distribution
supply = output.pivot(index='ID', columns='year', values='margin')

#coverage.head(n=5)

plt.figure()

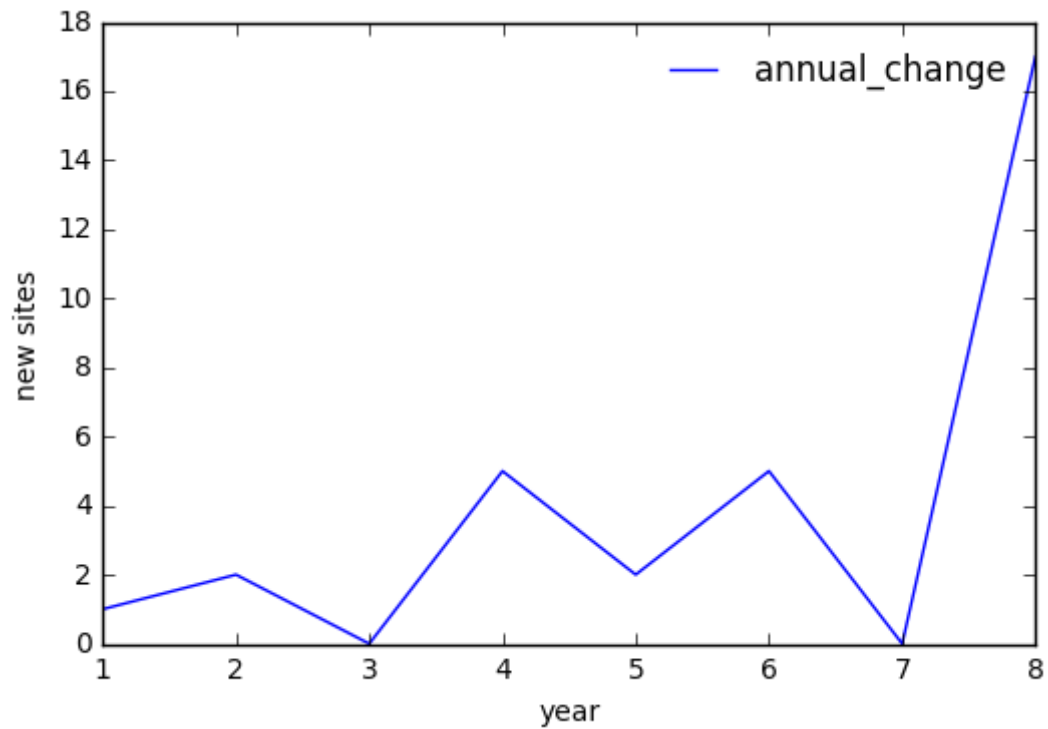
dp = supply.boxplot()
```



## Annual change in new sites

```
In [34]: plt.plot(cap_margin.year, cap_margin.annual_change)
plt.xlabel('year')
plt.ylabel('new sites')
plt.legend(framealpha=0, frameon=False)
```

Out[34]: <matplotlib.legend.Legend at 0xadbd5f8>



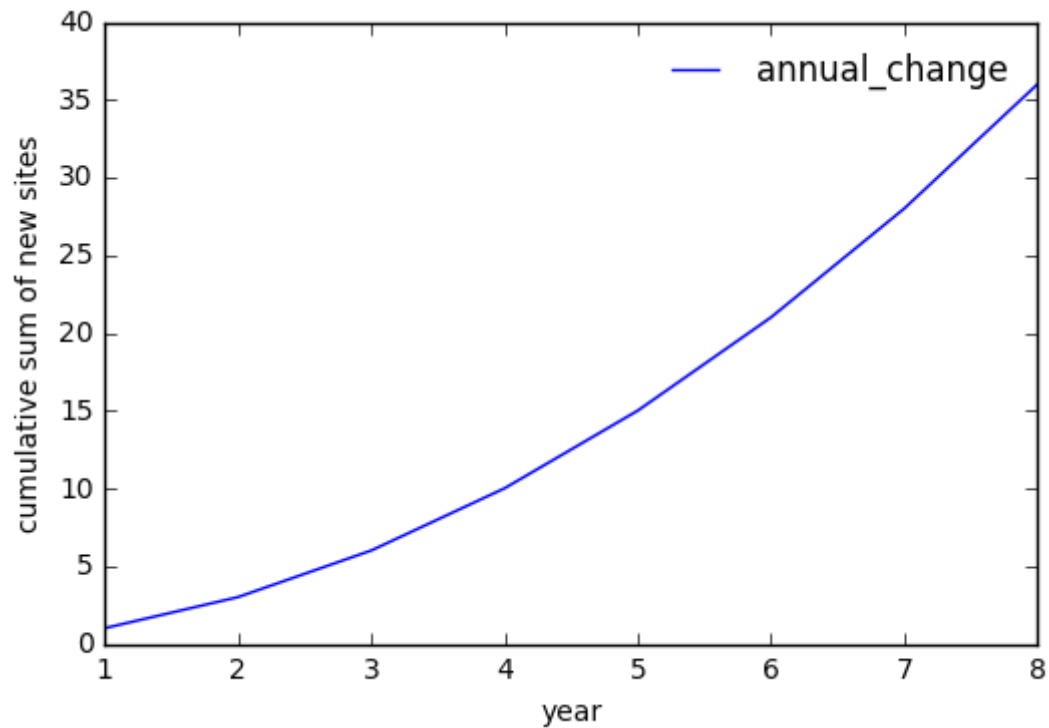
```
In [35]: ## Cumulative sum of new sites over time
```

```
In [36]: #sum by year
cap_margin['annual_change'] = cap_margin.cumsum()

#.groupby(['ID', 'year']).cumsum().groupby(level=[0]).cumsum()
#reset_index()
cap_margin.head(n=10)

plt.plot(cap_margin.year, cap_margin.annual_change)
plt.xlabel('year')
plt.ylabel('cumulative sum of new sites')
plt.legend(framealpha=0, frameon=False)
```

Out[36]: <matplotlib.legend.Legend at 0x9665eb8>

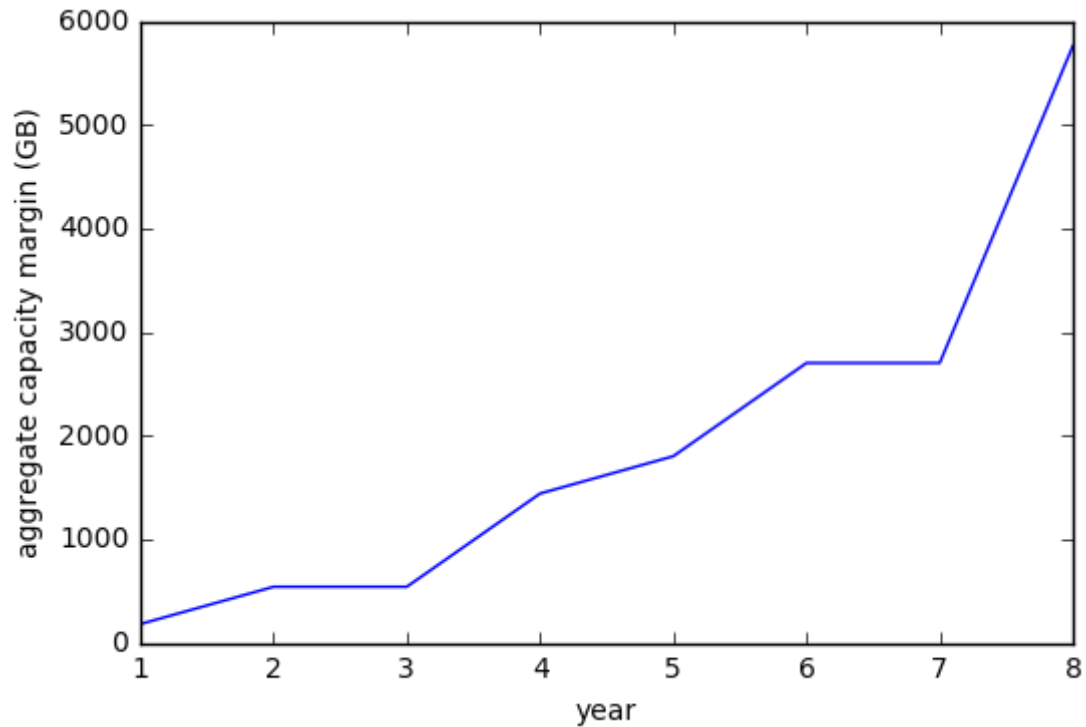


## Cumulative Total Cost of Ownership (TCO)



```
In [37]: #cumulative cost
plt.plot(cap_margin.year, cap_margin.TCO)
plt.xlabel('year')
plt.ylabel('aggregate capacity margin (GB)')
#plt.legend(framealpha=0, frameon=False)
```

Out[37]: <matplotlib.text.Text at 0xae23da0>

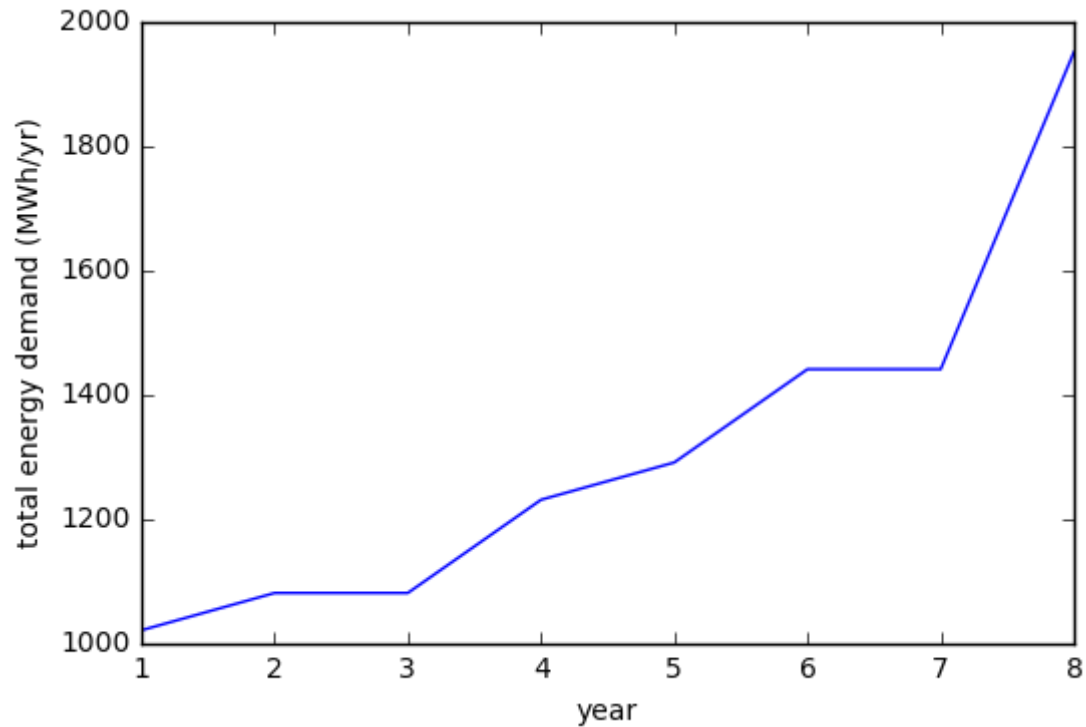


## Calculate energy demand of new assets

Importantly, this is purely illustrative for now, and does not include decommission of existing assets which is really important, so in reality this will be much lower.

```
In [38]: #energy demand
plt.plot(cap_margin.year, cap_margin.total_energy_demand)
plt.xlabel('year')
plt.ylabel('total energy demand (MWh/yr)')
#plt.legend(framealpha=0, frameon=False)
```

Out[38]: <matplotlib.text.Text at 0xaf22f98>



## Next steps

There are a number of next steps for the model, including:

- *Generating multiple scenarios*
- *Interventions including multiple technologies*
- *Dealing with decommissioning of assets*
- *Populating the demand and supply parts of the model with real data*
- *Populating the supply, cost and energy lookup tables with real data*
- *Dealing with decommissioning of assets*