

Modeling and Simulation in Python

Chapter 10

Copyright 2017 Allen Downey

License: [Creative Commons Attribution 4.0 International](#)

```
In [1]: # Configure Jupyter so figures appear in the notebook
%matplotlib inline

# Configure Jupyter to display the assigned value after an assignment
%config InteractiveShell.ast_node_interactivity='last_expr_or_assign'

# import functions from the modsim.py module
from modsim import *

from pandas import read_html
```

Under the hood

To get a `DataFrame` and a `Series`, I'll read the world population data and select a column.

`DataFrame` and `Series` contain a variable called `shape` that indicates the number of rows and columns.

```
In [2]: filename = 'data/World_population_estimates.html'
tables = read_html(filename, header=0, index_col=0, decimal='M')
table2 = tables[2]
table2.columns = ['census', 'prb', 'un', 'maddison',
                  'hyde', 'tanton', 'biraben', 'mj',
                  'thomlinson', 'durand', 'clark']
table2.shape
```

```
Out[2]: (67, 11)
```

```
In [3]: census = table2.census / 1e9
census.shape
```

```
Out[3]: (67,)
```

```
In [4]: un = table2.un / 1e9
un.shape
```

```
Out[4]: (67,)
```

A `DataFrame` contains `index`, which labels the rows. It is an `Int64Index`, which is similar to a NumPy array.

```
In [5]: table2.index
```

```
Out[5]: Int64Index([1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960,
                   1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971,
```

```
1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982,
1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993,
1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004,
2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015,
2016],
dtype='int64', name='Year')
```

And `columns`, which labels the columns.

```
In [6]: table2.columns
```

```
Out[6]: Index(['census', 'prb', 'un', 'maddison', 'hyde', 'tanton', 'biraben', 'mj',
               'thomlinson', 'durand', 'clark'],
              dtype='object')
```

And `values`, which is an array of values.

```
In [7]: table2.values
```

```
Out[7]: array([[2557628654, 2516000000.0, 2525149000.0, 2544000000.0,
                2527960000.0, 2400000000.0, 2527000000.0, 2500000000.0,
                2400000000.0, nan, 2486000000.0],
               [2594939877, nan, 2572850917.0, 2571663000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [2636772306, nan, 2619292068.0, 2617949000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [2682053389, nan, 2665865392.0, 2665959000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [2730228104, nan, 2713172027.0, 2716927000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [2782098943, nan, 2761650981.0, 2769074000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [2835299673, nan, 2811572031.0, 2822502000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [2891349717, nan, 2863042795.0, 2879934000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [2948137248, nan, 2916030167.0, 2939254000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [3000716593, nan, 2970395814.0, 2995909000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [3043001508, nan, 3026002942.0, 3041507000.0, 3042000000.0, nan,
                nan, nan, nan, nan],
               [3083966929, nan, 3082830266.0, 3082161000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [3140093217, nan, 3141071531.0, 3135787000.0, nan, nan, nan, nan,
                nan, nan, 3036000000.0],
               [3209827882, nan, 3201178277.0, 3201354000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [3281201306, nan, 3263738832.0, 3266477000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [3350425793, nan, 3329122479.0, 3333138000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [3420677923, nan, 3397475247.0, 3402224000.0, nan, nan, nan, nan,
                nan, nan, 3288000000.0],
               [3490333715, nan, 3468521724.0, 3471464000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [3562313822, nan, 3541674891.0, 3543086000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [3637159050, nan, 3616108749.0, 3615743000.0, nan, nan, nan, nan,
                nan, nan, nan],
               [3712697742, nan, 3691172616.0, 3691157000.0, 3710000000.0, nan,
                3637000000.0, nan, 3600000000.0, '3,600,000,000- 3,700,000,000',
                3632000000.0],
               [3790326948, nan, 3766754345.0, 3769818000.0, nan, nan, nan, nan,
```

nan, nan, nan],
[3866568653, nan, 3842873611.0, 3846499000.0, nan, nan, nan, nan,
nan, nan, nan],
[3942096442, nan, 3919182332.0, 3922793000.0, 3923000000.0, nan,
nan, nan, nan, nan, 3860000000.0],
[4016608813, nan, 3995304922.0, 3997677000.0, nan, nan, nan, nan,
nan, nan, nan],
[4089083233, nan, 4071020434.0, 4070671000.0, nan, nan, nan,
3900000000.0, 400000000.0, nan, nan],
[4160185010, nan, 4146135850.0, 4141445000.0, nan, nan, nan, nan,
nan, nan, nan],
[4232084578, nan, 4220816737.0, 4213539000.0, nan, nan, nan, nan,
nan, nan, nan],
[4304105753, nan, 4295664825.0, 4286317000.0, nan, nan, nan, nan,
nan, nan, nan],
[4379013942, nan, 4371527871.0, 4363144000.0, nan, nan, nan, nan,
nan, nan, nan],
[4451362735, nan, 4449048798.0, 4439529000.0, 4461000000.0, nan,
nan, nan, nan, nan, nan],
[4534410125, nan, 4528234634.0, 4514838000.0, nan, nan, nan, nan,
nan, nan, nan],
[4614566561, nan, 4608962418.0, 4587307000.0, nan, nan, nan, nan,
nan, nan, nan],
[4695736743, nan, 4691559840.0, 4676388000.0, nan, nan, nan, nan,
nan, nan, nan],
[4774569391, nan, 4776392828.0, 4756521000.0, nan, nan, nan, nan,
nan, nan, nan],
[4856462699, nan, 4863601517.0, 4837719000.0, nan, 5000000000.0,
nan, nan, nan, nan, nan],
[4940571232, nan, 4953376710.0, 4920968000.0, nan, nan, nan, nan,
nan, nan, nan],
[5027200492, nan, 5045315871.0, 5006672000.0, nan, nan, nan, nan,
nan, nan, nan],
[5114557167, nan, 5138214688.0, 5093306000.0, nan, nan, nan, nan,
nan, nan, nan],
[5201440110, nan, 5230000000.0, 5180540000.0, nan, nan, nan, nan,
nan, nan, nan],
[5288955934, nan, 5320816667.0, 5269029000.0, 5308000000.0, nan,
nan, nan, nan, nan, nan],
[5371585922, nan, 5408908724.0, 5351922000.0, nan, nan, nan, nan,
nan, nan, nan],
[5456136278, nan, 5494899570.0, 5435722000.0, nan, nan, nan, nan,
nan, nan, nan],
[5538268316, nan, 5578865109.0, 5518127000.0, nan, nan, nan, nan,
nan, nan, nan],
[5618682132, nan, 5661086346.0, 5599396000.0, nan, nan, nan, nan,
nan, nan, nan],
[5699202985, 5760000000.0, 5741822412.0, 5681575000.0, nan, nan,
nan, nan, nan, nan, nan],
[5779440593, nan, 5821016750.0, 5762212000.0, nan, nan, nan, nan,
nan, nan, nan],
[5857972543, 5840000000.0, 5898688337.0, 5842122000.0, nan, nan,
nan, nan, nan, nan, nan],
[5935213248, nan, 5975303657.0, 5921366000.0, nan, nan, nan, nan,
nan, nan, nan],
[6012074922, nan, 6051478010.0, 5999622000.0, nan, nan, nan, nan,
nan, nan, nan],
[6088571383, 6067000000.0, 6127700428.0, 6076558000.0,
6145000000.0, nan, nan, 5750000000.0, nan, nan, nan, nan],
[6165219247, 6137000000.0, 6204147026.0, 6154791000.0, nan, nan,
nan, nan, nan, nan, nan],
[6242016348, 6215000000.0, 6280853817.0, 6231704000.0, nan, nan,
nan, nan, nan, nan, nan],
[6318590956, 6314000000.0, 6357991749.0, 6308364000.0, nan, nan,
nan, nan, nan, nan, nan],

```
[6395699509, 6396000000.0, 6435705595.0, 6374056000.0, nan, nan,
nan, nan, nan, nan, nan],
[6473044732, 6477000000.0, 6514094605.0, 6462987000.0, nan, nan,
nan, nan, nan, nan, nan],
[6551263534, 6555000000.0, 6593227977.0, 6540214000.0, nan, nan,
nan, nan, nan, nan, nan],
[6629913759, 6625000000.0, 6673105937.0, 6616689000.0, nan, nan,
nan, nan, nan, nan, nan],
[6709049780, 6705000000.0, 6753649228.0, 6694832000.0, nan, nan,
nan, nan, nan, nan, nan],
[6788214394, 6809972000.0, 6834721933.0, 6764086000.0, nan, nan,
nan, nan, nan, nan, nan],
[6858584755, 6892319000.0, 6916183482.0, nan, nan, nan, nan, nan,
nan, nan, nan, nan],
[6935999491, 6986951000.0, 6997998760.0, nan, nan, nan, nan, nan,
nan, nan, nan, nan],
[7013871313, 7057075000.0, 7080072417.0, nan, nan, nan, nan, nan,
nan, nan, nan, nan],
[7092128094, 7136796000.0, 7162119434.0, nan, nan, nan, nan, nan,
nan, nan, nan, nan],
[7169968185, 7238184000.0, 7243784000.0, nan, nan, nan, nan, nan,
nan, nan, nan, nan],
[7247892788, 7336435000.0, 7349472000.0, nan, nan, nan, nan, nan,
nan, nan, nan, nan],
[7325996709, 7418151841.0, nan, nan, nan, nan, nan, nan, nan, nan,
nan]], dtype=object)
```

A Series does not have columns , but it does have name .

```
In [8]: census.name
```

```
Out[8]: 'census'
```

It contains values , which is an array.

```
In [9]: census.values
```

```
Out[9]: array([2.55762865, 2.59493988, 2.63677231, 2.68205339, 2.7302281 ,
2.78209894, 2.83529967, 2.89134972, 2.94813725, 3.00071659,
3.04300151, 3.08396693, 3.14009322, 3.20982788, 3.28120131,
3.35042579, 3.42067792, 3.49033371, 3.56231382, 3.63715905,
3.71269774, 3.79032695, 3.86656865, 3.94209644, 4.01660881,
4.08908323, 4.16018501, 4.23208458, 4.30410575, 4.37901394,
4.45136274, 4.53441012, 4.61456656, 4.69573674, 4.77456939,
4.8564627 , 4.94057123, 5.02720049, 5.11455717, 5.20144011,
5.28895593, 5.37158592, 5.45613628, 5.53826832, 5.61868213,
5.69920299, 5.77944059, 5.85797254, 5.93521325, 6.01207492,
6.08857138, 6.16521925, 6.24201635, 6.31859096, 6.39569951,
6.47304473, 6.55126353, 6.62991376, 6.70904978, 6.78821439,
6.85858475, 6.93599949, 7.01387131, 7.09212809, 7.16996819,
7.24789279, 7.32599671])
```

And it contains index :

```
In [10]: census.index
```

```
Out[10]: Int64Index([1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960,
1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971,
1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982,
1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993,
1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004,
2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015,
2016],  
dtype='int64', name='Year')
```

If you ever wonder what kind of object a variable refers to, you can use the `type` function. The result indicates what type the object is, and the module where that type is defined.

`DataFrame`, `Int64Index`, `Index`, and `Series` are defined by Pandas.

`ndarray` is defined by NumPy.

```
In [11]: type(table2)
```

```
Out[11]: pandas.core.frame.DataFrame
```

```
In [12]: type(table2.index)
```

```
Out[12]: pandas.core.indexes.numeric.Int64Index
```

```
In [13]: type(table2.columns)
```

```
Out[13]: pandas.core.indexes.base.Index
```

```
In [14]: type(table2.values)
```

```
Out[14]: numpy.ndarray
```

```
In [15]: type(census)
```

```
Out[15]: pandas.core.series.Series
```

```
In [16]: type(census.index)
```

```
Out[16]: pandas.core.indexes.numeric.Int64Index
```

```
In [17]: type(census.values)
```

```
Out[17]: numpy.ndarray
```

Optional exercise

The following exercise provides a chance to practice what you have learned so far, and maybe develop a different growth model. If you feel comfortable with what we have done so far, you might want to give it a try.

Optional Exercise: On the Wikipedia page about world population estimates, the first table contains estimates for prehistoric populations. The following cells process this table and plot some of the results.

```
In [18]: filename = 'data/World_population_estimates.html'  
tables = read_html(filename, header=0, index_col=0, decimal='M')  
len(tables)
```

```
Out[18]: 6
```

Select `tables[1]` , which is the second table on the page.

```
In [19]: table1 = tables[1]
table1.head()
```

Out[19] :

Population Reference Bureau (1973–2016)[15]	United Nations Department of Economic and Social Affairs (2015)[16]	Maddison (2008) [17]	HYDE (2010) [citation needed]	Tanton (1994) [18]	Biraben (1980) [19]	McEvedy & Jones (1978) [20]	Thomlinson (1975)[21]	Durand (1974) [22]
Year								
-10000	NaN	NaN	NaN	2M[24]	NaN	NaN	4.0	1–10M
-9000	NaN	NaN	NaN	4.	NaN	NaN	NaN	NaN
-8000	5.	NaN	NaN	5.	NaN	NaN	NaN	5–10M
-7000	NaN	NaN	NaN	8.	NaN	NaN	NaN	NaN
-6000	NaN	NaN	NaN	11.	NaN	NaN	NaN	NaN

Not all agencies and researchers provided estimates for the same dates. Again `NaN` is the special value that indicates missing data.

```
In [20]: table1.tail()
```

Out[20] :

Population Reference Bureau (1973–2016)[15]	United Nations Department of Economic and Social Affairs (2015)[16]	Maddison (2008) [17]	HYDE (2010) [citation needed]	Tanton (1994) [18]	Biraben (1980) [19]	McEvedy & Jones (1978) [20]	Thomlinson (1975)[21]	Durand (1974) [22]
Year								
1913	NaN	NaN	1793.	NaN	NaN	NaN	NaN	NaN
1920	NaN	1860.0	1863.	1912.	NaN	NaN	NaN	NaN
1925	NaN	NaN	NaN	NaN	NaN	NaN	2000.0	NaN
1930	NaN	2070.0	NaN	2092.	NaN	NaN	NaN	NaN
1940	NaN	2300.0	2299.	2307.	NaN	NaN	NaN	NaN

Again, we'll replace the long column names with more convenient abbreviations.

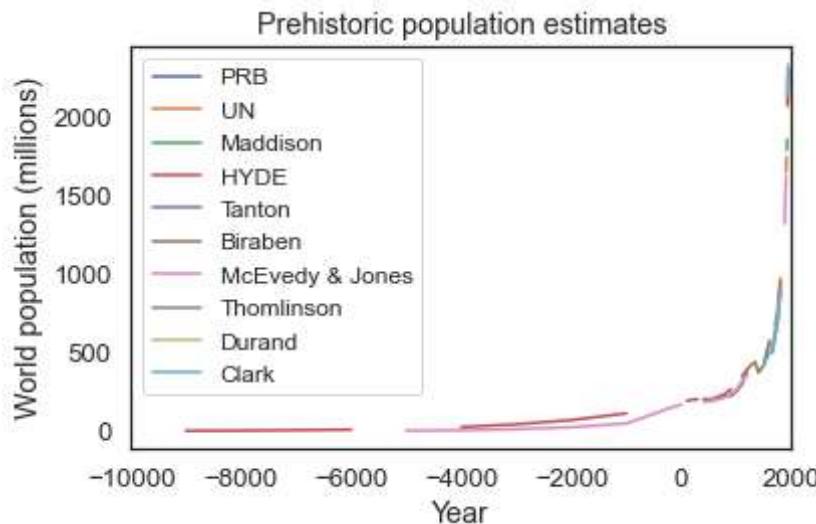
```
In [21]: table1.columns = ['PRB', 'UN', 'Maddison', 'HYDE', 'Tanton',
'Biraben', 'McEvedy & Jones', 'Thomlinson', 'Durand', 'Clark']
```

Some of the estimates are in a form Pandas doesn't recognize as numbers, but we can coerce them to be numeric.

```
In [22]: for col in table1.columns:  
    table1[col] = pd.to_numeric(table1[col], errors='coerce')
```

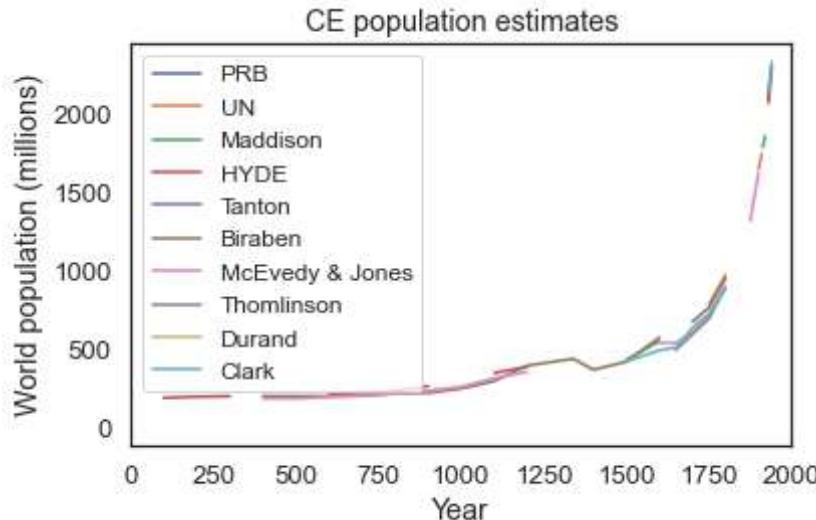
Here are the results. Notice that we are working in millions now, not billions.

```
In [23]: table1.plot()  
decorate(xlim=[-10000, 2000], xlabel='Year',  
        ylabel='World population (millions)',  
        title='Prehistoric population estimates')  
plt.legend(fontsize='small');
```



We can use `xlim` to zoom in on everything after Year 0.

```
In [24]: table1.plot()  
decorate(xlim=[0, 2000], xlabel='Year',  
        ylabel='World population (millions)',  
        title='CE population estimates')  
plt.legend(fontsize='small');
```



See if you can find a model that fits these data well from Year 0 to 1950.

How well does your best model predict actual population growth from 1950 to the present?

```
In [25]: HYDE=table1.HYDE  
print(HYDE)
```

```
Year  
-10000      NaN  
-9000       4.0  
-8000       5.0  
-7000       8.0  
-6000      11.0  
-5000      NaN  
-4000      28.0  
-3000      45.0  
-2000      72.0  
-1000     115.0  
-200      NaN  
1          NaN  
100      195.0  
200      202.0  
300      205.0  
350      NaN  
400      209.0  
500      NaN  
600      213.0  
700      226.0  
800      240.0  
900      269.0  
1000     NaN  
1100     353.0  
1200     393.0  
1250     NaN  
1300     392.0  
1340     NaN  
1400     390.0  
1500     NaN  
1600     NaN  
1650     NaN  
1700     NaN  
1750    814.0  
1800     NaN  
1820     NaN  
1850   1263.0  
1870     NaN  
1875     NaN  
1900     NaN  
1910   1777.0  
1913     NaN  
1920   1912.0  
1925     NaN  
1930   2092.0  
1940   2307.0  
Name: HYDE, dtype: float64
```

```
In [26]: x=HYDE.index[11:]  
print (x)
```

```
Int64Index([ 1, 100, 200, 300, 350, 400, 500, 600, 700, 800, 900,  
1000, 1100, 1200, 1250, 1300, 1340, 1400, 1500, 1600, 1650, 1700,  
1750, 1800, 1820, 1850, 1870, 1875, 1900, 1910, 1913, 1920, 1925,  
1930, 1940],  
dtype='int64', name='Year')
```

```
In [27]: # Solution goes here
```

```
#find a model that fits these data well from Year 0 to 1950.
```

```
x = linspace(100, 1950)
print(x)
```

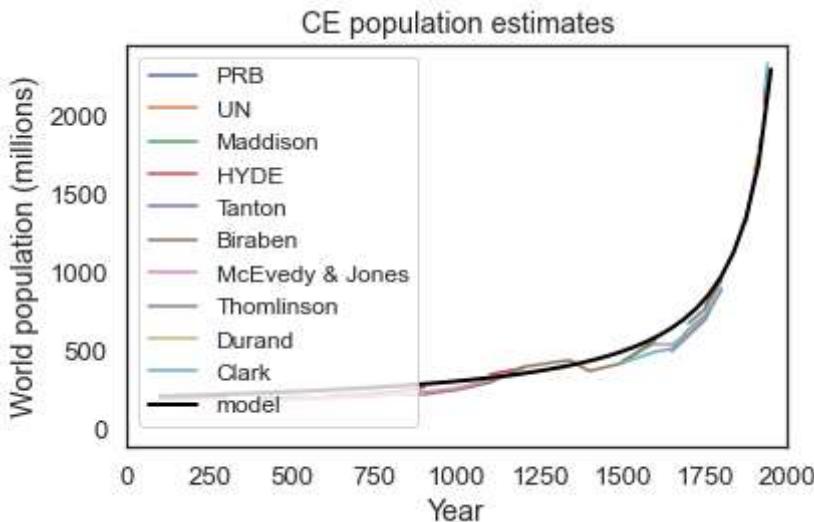
```
[ 100.          137.75510204  175.51020408  213.26530612  251.02040816
 288.7755102   326.53061224  364.28571429  402.04081633  439.79591837
 477.55102041  515.30612245  553.06122449  590.81632653  628.57142857
 666.32653061  704.08163265  741.83673469  779.59183673  817.34693878
 855.10204082  892.85714286  930.6122449   968.36734694  1006.12244898
1043.87755102 1081.63265306 1119.3877551  1157.14285714 1194.89795918
1232.65306122 1270.40816327 1308.16326531 1345.91836735 1383.67346939
1421.42857143 1459.18367347 1496.93877551 1534.69387755 1572.44897959
1610.20408163 1647.95918367 1685.71428571 1723.46938776 1761.2244898
1798.97959184 1836.73469388 1874.48979592 1912.24489796 1950. ]
```

```
In [28]: y = 100 + 220000 / (2050 - x)
print(y)
```

```
[ 212.82051282  215.04802561  217.36526946  219.77777778  222.2915485
 224.91309386  227.64949674  230.50847458  233.49845201  236.62864385
 239.9091499   243.35106383  246.9665985   250.76923077  254.77386935
 258.99705015  263.45716452  268.17472699  273.17269076  278.47682119
 284.11614005  290.12345679  296.53600729  303.39622642  310.75268817
 318.66125761  327.18651212  336.40350877  346.4       357.27923628
 369.16354557  382.19895288  396.56121045  412.46376812  430.16845329
 450.          472.36614853  497.78597786  526.93069307  560.68376068
 600.23201856  647.20812183  703.92156863  773.75      861.83745583
 976.42276423 1131.57894737 1353.48837209 1697.03703704 2300. ]
```

```
In [29]: table1.plot()
plot(x, y, color = 'black', label = 'model')

decorate(xlim = [0, 2000], xlabel = 'Year',
         ylabel = 'World population (millions)',
         title = 'CE population estimates')
plt.legend(fontsize = 'small');
```



```
In [30]: #How well does your best model predict actual population growth from 1950 to the present
# Solution goes here
```

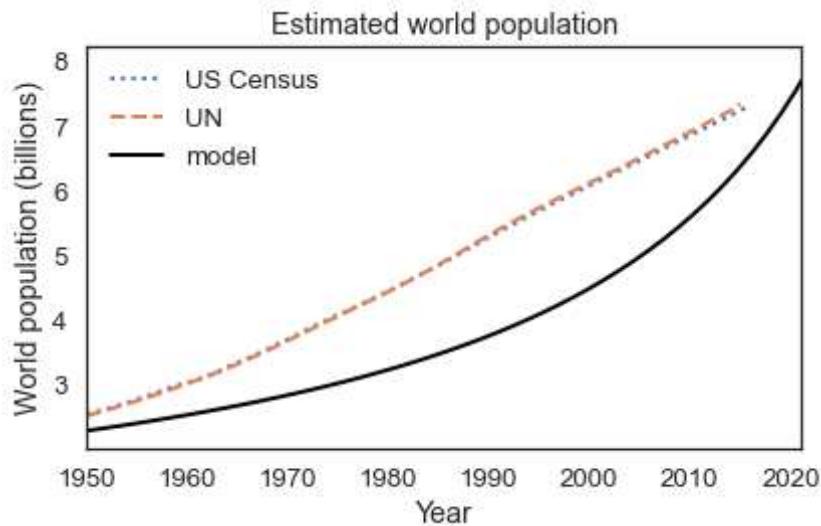
```
plot(census, ':', label = 'US Census')
plot(un, '--', label = 'UN')
```

```

x = linspace(1950, 2022)
y = 100 + 220000 / (2050 - x)
plot(x, y / 1000, color = 'black', label = 'model')

decorate(xlim = [1950, 2021], xlabel = 'Year',
         ylabel = 'World population (billions)',
         title = 'Estimated world population')

```



The model is underestimating the population growth in the interval, however, towards the endpoint will the model is overestimating. Looks like a linear fit or straight line would be better for this interval.