Trend Analysis of the World's Population, Poverty and Fertility Rates, and Child Mortality

October 18, 2020

Student: Odaiclet Piccinini

[88]:
```
# Initialize OK
from client.api.notebook import Notebook
ok = Notebook('project1.ok')
```

```
=====================================================================
Assignment: Project 1: World Progress
OK, version v1.18.1
=====================================================================
```

0.1 World Progress

In this project, you'll explore data from Gapminder.org, a website dedicated to providing a fact-based view of the world and how it has changed. That site includes several data visualizations and presentations, but also publishes the raw data that we will use in this project to recreate and extend some of their most famous visualizations.

The Gapminder website collects data from many sources and compiles them into tables that describe many countries around the world. All of the data they aggregate are published in the Systema Globalis. Their goal is "to compile all public statistics; Social, Economic and Environmental; into a comparable total dataset." All data sets in this project are copied directly from the Systema Globalis without any changes.

This project is dedicated to Hans Rosling (1948-2017), who championed the use of data to understand and prioritize global development challenges.

[89]:
```
from datascience import *
import numpy as np

%matplotlib inline
import matplotlib.pyplot as plots
plots.style.use('fivethirtyeight')

from client.api.notebook import Notebook
ok = Notebook('project1.ok')
```

```
=====================================================================
Assignment: Project 1: World Progress
OK, version v1.18.1
=====================================================================
```

0.2 1. Global Population Growth

The global population of humans reached 1 billion around 1800, 3 billion around 1960, and 7 billion around 2011. The potential impact of exponential population growth has concerned scientists, economists, and politicians alike.

The UN Population Division estimates that the world population will likely continue to grow throughout the 21st century, but at a slower rate, perhaps reaching 11 billion by 2100. However, the UN does not rule out scenarios of more extreme growth.

In this section, we will examine some of the factors that influence population growth and how they are changing around the world.

The first table we will consider is the total population of each country over time. Run the cell below.

[91]:
```
population = Table.read_table('population.csv').where("time", are.below(2021))
population.show(50)
```

```
<IPython.core.display.HTML object>
```

Note: The population csv file can also be found here. The data for this project was downloaded in April 2020.

0.2.1 Haiti

In the `population` table, the `geo` column contains three-letter codes established by the International Organization for Standardization (ISO) in the Alpha-3 standard. We will begin by taking a close look at Haiti. Use the ISO link to find the 3-letter code for Haiti.

Question 1. Create a table called `h_pop` that has two columns labeled `time` and `population_total`. The first column should contain the years from 1970 through 2020 (including both 1970 and 2020) and the second should contain the population of Haiti in each of those years.

[92]:
```
h_pop = Table().with_columns(
    "time", population.where('geo', are.equal_to('hti')).where('time', are.
↪between(1970, 2021)).column('time'),
    "population_total", population.where('geo', are.equal_to('hti')).
↪where('time', are.between(1970, 2021)).column('population_total')
)
h_pop.show(4)
```

```
<IPython.core.display.HTML object>
```

[93]:
```
ok.grade("q1_1");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

Run the following cell to create a table called `h_decade` that has the population of Haiti for every ten years starting in 1970 and going up to 2020. At a glance, it appears that the population of Haiti has been growing quickly indeed!

[94]:
```
h_pop.set_format('population_total', NumberFormatter)

tens = np.arange(1970, 2021, 10) # 1970, 1980, 1990, ...
h_decade = h_pop.sort('time').where('time', are.contained_in(tens))
h_decade.show()
```

```
<IPython.core.display.HTML object>
```

Question 2. Assign `initial` to an array that contains the population for every ten year interval from 1970 to 2010. Then, assign `changed` to an array that contains the population for every ten year interval from 1980 to 2020. You should use the `h_decade` table to create both arrays, first filtering the table to only contain the relevant years.

The annual growth rate for a time period is equal to:

$$\left(\frac{\text{Population at end of period}}{\text{Population at start of period}}^{\frac{1}{\text{number of years}}}\right) - 1$$

We have provided the code below that uses `initial` and `changed` in order to add a column to `h_decade` called `annual_growth`. Don't worry about the calculation of the growth rates; run the test below to test your solution.

If you are interested in how we came up with the formula for growth rates, consult the growth rates section of the textbook.

[95]:
```
initial = h_decade.where('time', are.between(1970, 2011)).
 ↪column('population_total')
changed = h_decade.where('time', are.between(1980, 2021)).
 ↪column('population_total')
```

```
growth_rates = ((changed/initial)**0.1)-1

h_1970_through_2010 = h_decade.where('time', are.below_or_equal_to(2010))
h_decade_growth = h_1970_through_2010.with_column('annual_growth', growth_rates)
# Don't change this line!
h_decade_growth.set_format('annual_growth', PercentFormatter)
```

[95]:
```
time | population_total | annual_growth
1970 | 4,676,237        | 1.90%
1980 | 5,643,175        | 2.23%
1990 | 7,037,915        | 1.86%
2000 | 8,463,802        | 1.63%
2010 | 9,949,318        | 1.37%
```

[96]:
```
ok.grade("q1_2");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

While the population has grown every decade since 1970, the annual growth rate decreased dramatically from 1980 to 2020. Let's look at some other information in order to develop a possible explanation. Run the next cell to load three additional tables of measurements about countries over time.

[97]:
```
life_expectancy = Table.read_table('life_expectancy.csv').where('time', are.
 ↪below(2021))
child_mortality = Table.read_table('child_mortality.csv').relabel(2,␣
 ↪'child_mortality_under_5_per_1000_born').where('time', are.below(2021))
fertility = Table.read_table('fertility.csv').where('time', are.below(2021))
```

The `life_expectancy` table contains a statistic that is often used to measure how long people live, called *life expectancy at birth*. This number, for a country in a given year, does not measure how long babies born in that year are expected to live. Instead, it measures how long someone would live, on average, if the *mortality conditions* in that year persisted throughout their lifetime. These "mortality conditions" describe what fraction of people at each age survived the year. So, it is a way of measuring the proportion of people that are staying alive, aggregated over different age groups in the population.

Run the following cells below to see `life_expectancy`, `child_mortality`, and `fertility`. Refer back to these tables as they will be helpful for answering further questions!

[98]:
```
life_expectancy
```

[98]:
```
geo  | time | life_expectancy_years
afg  | 1800 | 28.21
afg  | 1801 | 28.2
afg  | 1802 | 28.19
afg  | 1803 | 28.18
afg  | 1804 | 28.17
afg  | 1805 | 28.16
afg  | 1806 | 28.15
afg  | 1807 | 28.14
afg  | 1808 | 28.13
afg  | 1809 | 28.12
… (41240 rows omitted)
```

[99]:
```
child_mortality
```

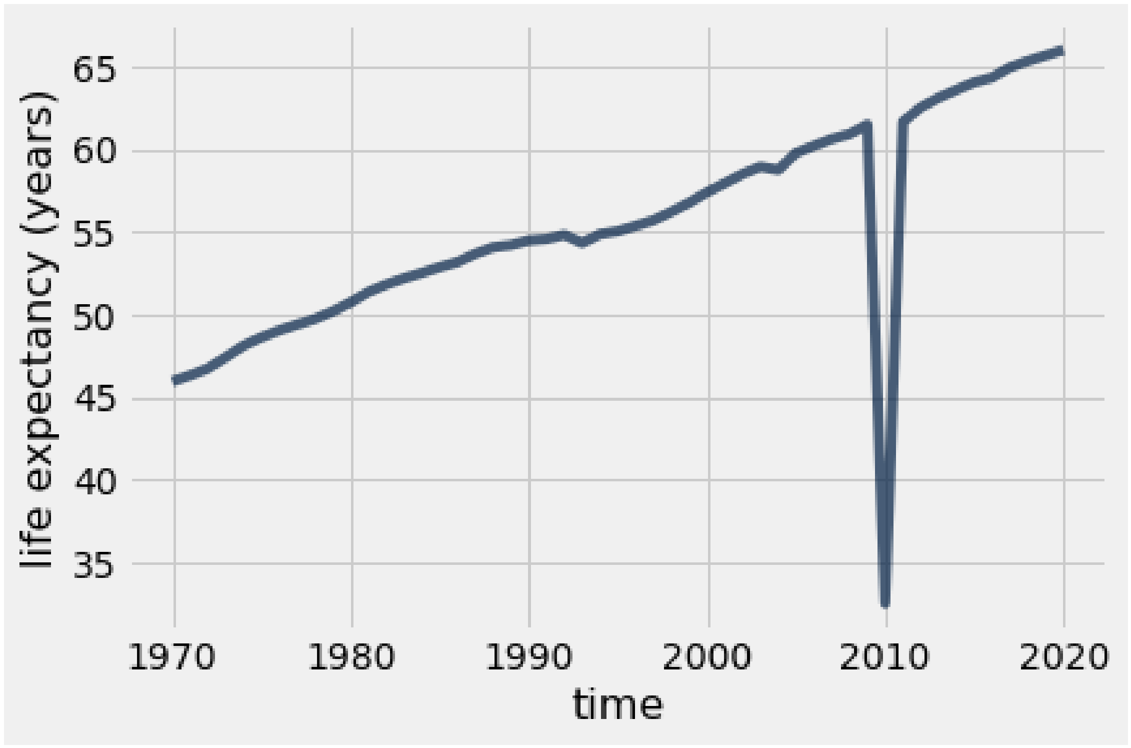
[99]:
```
geo  | time | child_mortality_under_5_per_1000_born
afg  | 1800 | 468.58
afg  | 1801 | 468.58
afg  | 1802 | 468.58
afg  | 1803 | 468.58
afg  | 1804 | 468.58
afg  | 1805 | 468.58
afg  | 1806 | 469.98
afg  | 1807 | 469.98
afg  | 1808 | 469.98
afg  | 1809 | 469.98
… (41727 rows omitted)
```

[100]:
```
fertility
```

[100]:
```
geo  | time | children_per_woman_total_fertility
abw  | 1800 | 5.64
abw  | 1801 | 5.64
abw  | 1802 | 5.64
abw  | 1803 | 5.64
abw  | 1804 | 5.64
abw  | 1805 | 5.64
abw  | 1806 | 5.64
abw  | 1807 | 5.64
abw  | 1808 | 5.64
abw  | 1809 | 5.64
… (44625 rows omitted)
```

Question 3. Perhaps population is growing more slowly because people aren't living as long. Use the `life_expectancy` table to draw a line graph with the years 1970 and later on the horizontal axis that shows how the *life expectancy at birth* has changed in Haiti.

[101]:
```
#Fill in code here
our_table = Table().with_columns("time", life_expectancy.where("geo", are.
↪equal_to("hti")).where("time", are.above_or_equal_to(1970)).column('time'),
                              "life expectancy (years)", life_expectancy.
↪where("geo", are.equal_to("hti")).where("time", are.above_or_equal_to(1970)).
↪column("life_expectancy_years")
                              )
our_table.plot("time", "life expectancy (years)")
```



Question 4. Assuming everything else stays the same, do the trends in life expectancy in the graph above directly explain why the population growth rate decreased from 1980 to 2020 in Haiti? Why or why not?

Hint: What happened in Haiti in 2010, and does that event explain the overall change in population growth rate?

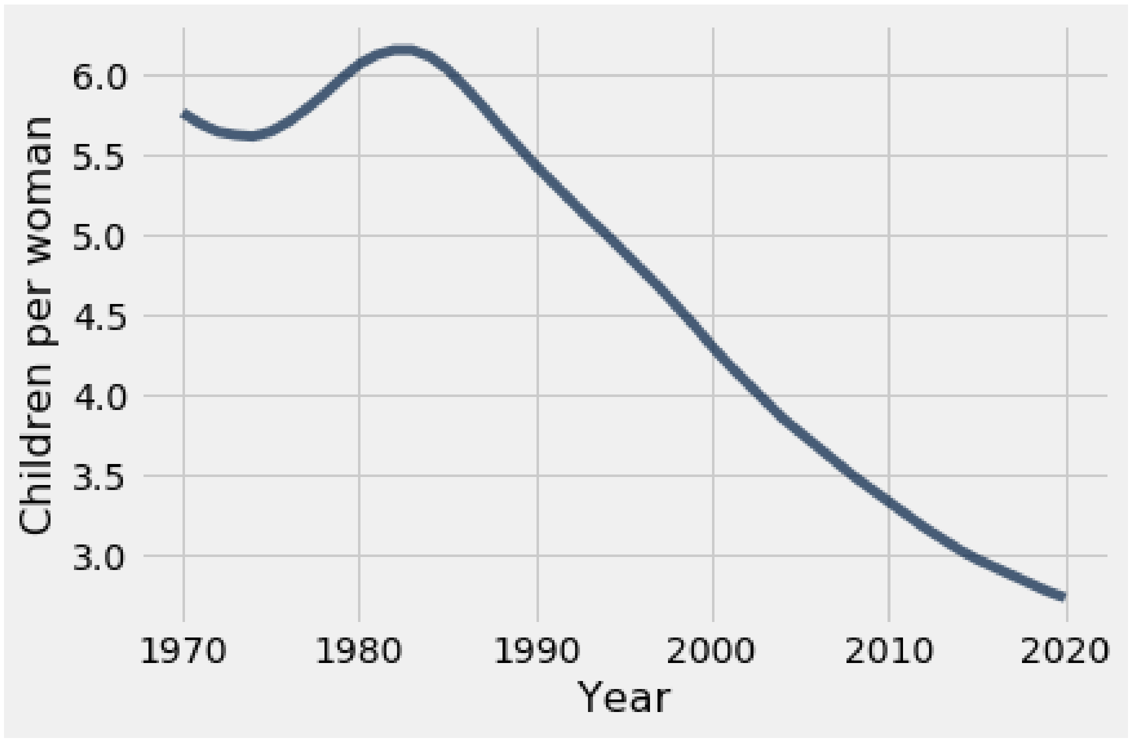
There is a strong association between the trends in life expectancy in the graph above but this does not directly explain why the population growth rate decreased from 1980 to 2020 in Haiti. Based on the graph, we are only able to see the how the life expectancy in Haiti increased over time. We are also able to visualize that during 2010 there was a strong decrease in life expectancy, which we attribute to the natural disaster that occurred in 2010 (An 8.0 scale earthquake). In order to visualize an direct explanation between the actual population growth rate in Haiti between 1980 and 2020, we will need data of the amount of people born in Haiti per year.

The `fertility` table contains a statistic that is often used to measure how many babies are being born, the *total fertility rate*. This number describes the number of children a woman would have in her lifetime, on average, if the current rates of birth by age of the mother persisted throughout her child bearing years, assuming she survived through age 49.

Question 5. Write a function `fertility_over_time` that takes the Alpha-3 code of a `country` and a `start` year. It returns a two-column table with labels `Year` and `Children per woman` that can be used to generate a line chart of the country's fertility rate each year, starting at the `start` year. The plot should include the `start` year and all later years that appear in the `fertility` table.

Then, in the next cell, call your `fertility_over_time` function on the Alpha-3 code for Haiti and the year 1970 in order to plot how Haiti's fertility rate has changed since 1970. Note that the function `fertility_over_time` should not return the plot itself. **The expression that draws the line plot is provided for you; please don't change it.**

[102]:
```
def fertility_over_time(country, start):
    """Create a two-column table that describes a country's total fertility␣
 ↪rate each year."""
    country_fertility = fertility.where("geo", are.equal_to(country)).
 ↪where('time', are.above_or_equal_to(start)).
 ↪column('children_per_woman_total_fertility')
    country_fertility_after_start = fertility.where("geo", are.
 ↪equal_to(country)).where('time', are.above_or_equal_to(start)).column('time')
    cleaned_table = Table().with_columns("Year", country_fertility_after_start,
                                        "Children per woman", country_fertility)
    return cleaned_table

haiti_code = 'hti'
fertility_over_time(haiti_code, 1970).plot(0, 1) # You should *not* change this␣
 ↪line.
```



[103]:
```
ok.grade("q1_5");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 6. Assuming everything else is constant, do the trends in fertility in the graph above help directly explain why the population growth rate decreased from 1980 to 2020 in Haiti? Why or why not?

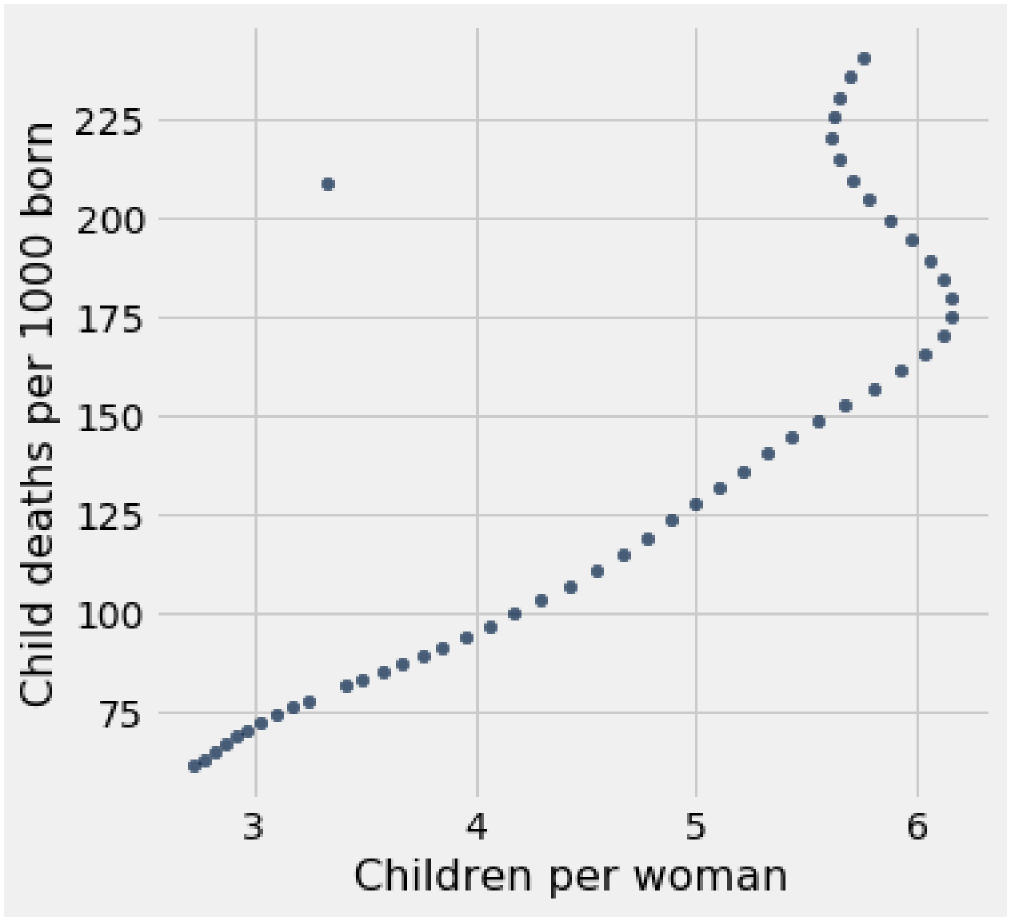
Assuming everything else is constant, the trends in fertility in the graph above do help directly explain why the population growth rate decreased from 1980 to 2020 in Haiti. Based on our graph, there is a drop in fertility which in turn directly explain the decrease in population growth.

It has been observed that lower fertility rates are often associated with lower child mortality rates. The link has been attributed to family planning: if parents can expect that their children will all survive into adulthood, then they will choose to have fewer children. We can see if this association is evident in Haiti by plotting the relationship between total fertility rate and child mortality rate per 1000 children.

Question 7. Using both the `fertility` and `child_mortality` tables, draw a scatter diagram that has Haiti's total fertility on the horizontal axis and its child mortality on the vertical axis with one point for each year, starting with 1970.

The expression that draws the scatter diagram is provided for you; please don't change it. Instead, create a table called `post_1969_fertility_and_child_mortality` with the appropriate column labels and data in order to generate the chart correctly. Use the label `Children per woman` to describe total fertility and the label `Child deaths per 1000 born` to describe child mortality.

[104]:
```
hti_fertility = fertility.where("geo", are.equal_to("hti"))
hti_child_mortality = child_mortality.where("geo", are.equal_to("hti"))
fertility_and_child_mortality = hti_fertility.join("time", hti_child_mortality)
post_1969_fertility_and_child_mortality = fertility_and_child_mortality.
 ↪where("time", are.above_or_equal_to(1970)).select(
    "time", "children_per_woman_total_fertility",␣
 ↪"child_mortality_under_5_per_1000_born").relabel(
    "children_per_woman_total_fertility", "Children per woman").
 ↪relabel("child_mortality_under_5_per_1000_born","Child deaths per 1000 born")
# Don't change this line!
post_1969_fertility_and_child_mortality.scatter('Children per woman', 'Child␣
 ↪deaths per 1000 born')
```



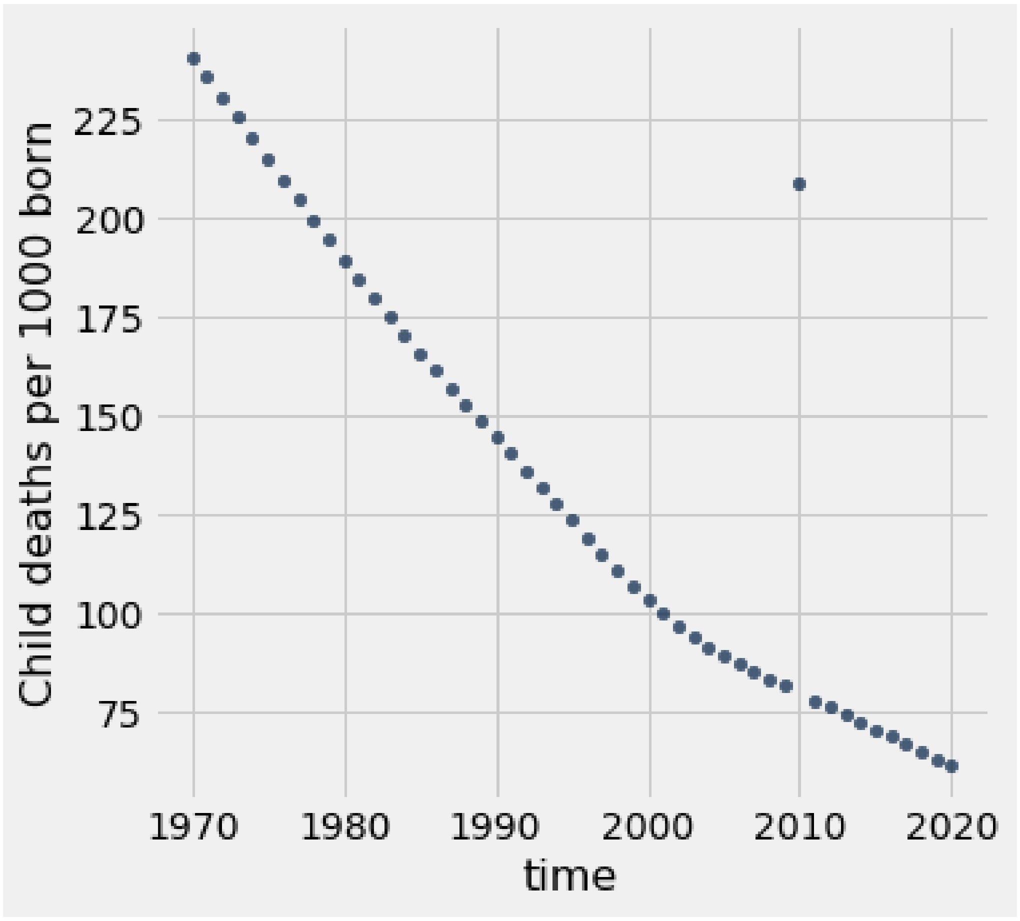
```
[105]: ok.grade("q1_7");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 8. In one or two sentences, describe the association (if any) that is illustrated by this scatter diagram. Does the diagram show that reduced child mortality causes parents to choose to have fewer children? Beyond the general association, are there any irregularities or odd points that would be worth investigating?

This scatter plot shows that there is a positive association between child deaths per 1000 born and the amount of children born per woman. As the children born per woman increases the child deaths per 1000 born also simultaneously increases; however, there is an irregularity present in the chart that is worth looking at. As depicted by the scatter plot, there is an outlier point above 200 child deaths vs. 3.33 children born per woman. We are unsure as to what is causing this outlier point, but in order to find out more information, we should look at the time in which this point occurred.

```
[106]: # Don't change this line!
       post_1969_fertility_and_child_mortality.scatter('time', 'Child deaths per 1000␣
        ↪born')
```



Question 9. Using our knowledge that historical factors may affect Haiti's population, we decide to plot the child mortality rate per 1000 born against time, as seen in the scatter plot above. The y-axis remains the same as the plot from 1.8, but we are now plotting against time rather than the number of children per woman. Does this new visualization give us a better understanding of the outlier in 1.8?

Yes, this new visualization gives us a better understanding of the outlier of 1.8. Since now we have time as our x axis we can infer that the outlier is attributed to the earthquake in Haiti that occurred in 2010.

0.2.2 Checkpoint (due Friday 9/25)

Congratulations, you have reached the checkpoint! Run the submit cell below to generate the checkpoint submission.

```
[107]: _ = ok.submit()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

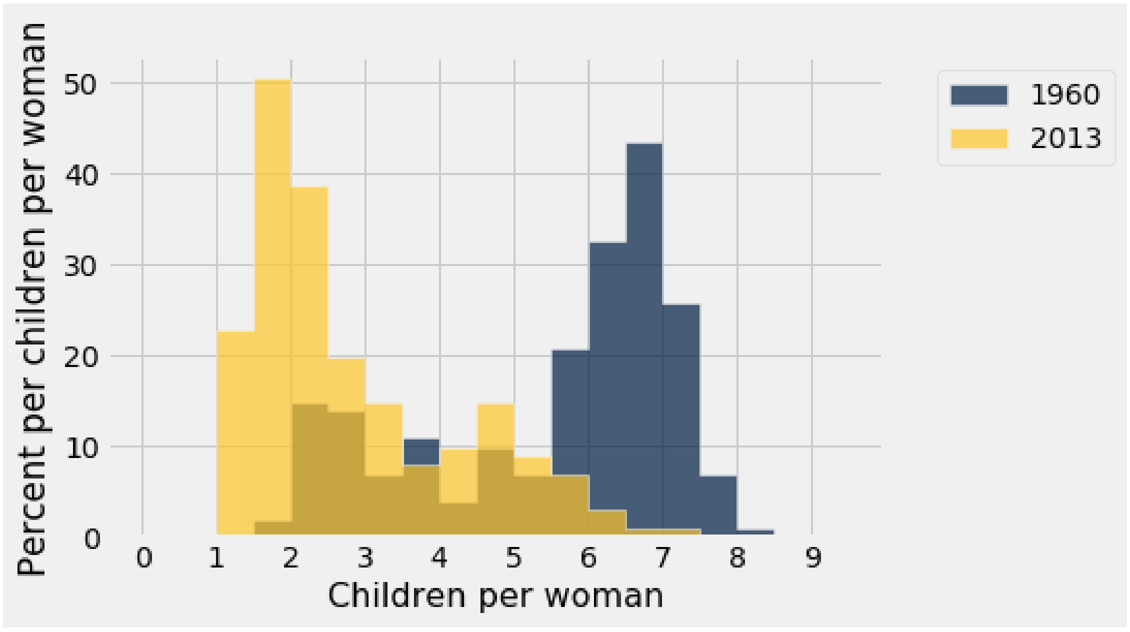
```
Saving notebook… Saved 'project1.ipynb'.
Submit… 100% complete
Submission successful for user: opiccinini2308@berkeley.edu
URL: https://okpy.org/cal/data8/fa20/project1/submissions/gnrXPY
```

0.2.3 The World

The change observed in Haiti since 1970 can also be observed in many other developing countries: health services improve, life expectancy increases, and child mortality decreases. At the same time, the fertility rate often plummets, and so the population growth rate decreases despite increasing longevity.

Run the cell below to generate two overlaid histograms, one for 1960 and one for 2013, that show the distributions of total fertility rates for these two years among all 201 countries in the `fertility` table.

[108]:
```
Table().with_columns(
    '1960', fertility.where('time', 1960).column(2),
    '2013', fertility.where('time', 2013).column(2)
).hist(bins=np.arange(0, 10, 0.5), unit='child per woman')
_ = plots.xlabel('Children per woman')
_ = plots.ylabel('Percent per children per woman')
_ = plots.xticks(np.arange(10))
```



Question 9. Assign `fertility_statements` to an array of the numbers of each statement below that can be correctly inferred from these histograms.

1. About the same number of countries had a fertility rate between 3.5 and 4.5 in both 1960 and 2013.
2. In 1960, less than 20% of countries had a fertility rate below 3.
3. At least half of countries had a fertility rate between 5 and 8 in 1960.
4. In 2013, about 40% of countries had a fertility rate between 1.5 and 2.
5. At least half of countries had a fertility rate below 3 in 2013.
6. More countries had a fertility rate above 3 in 1960 than in 2013.

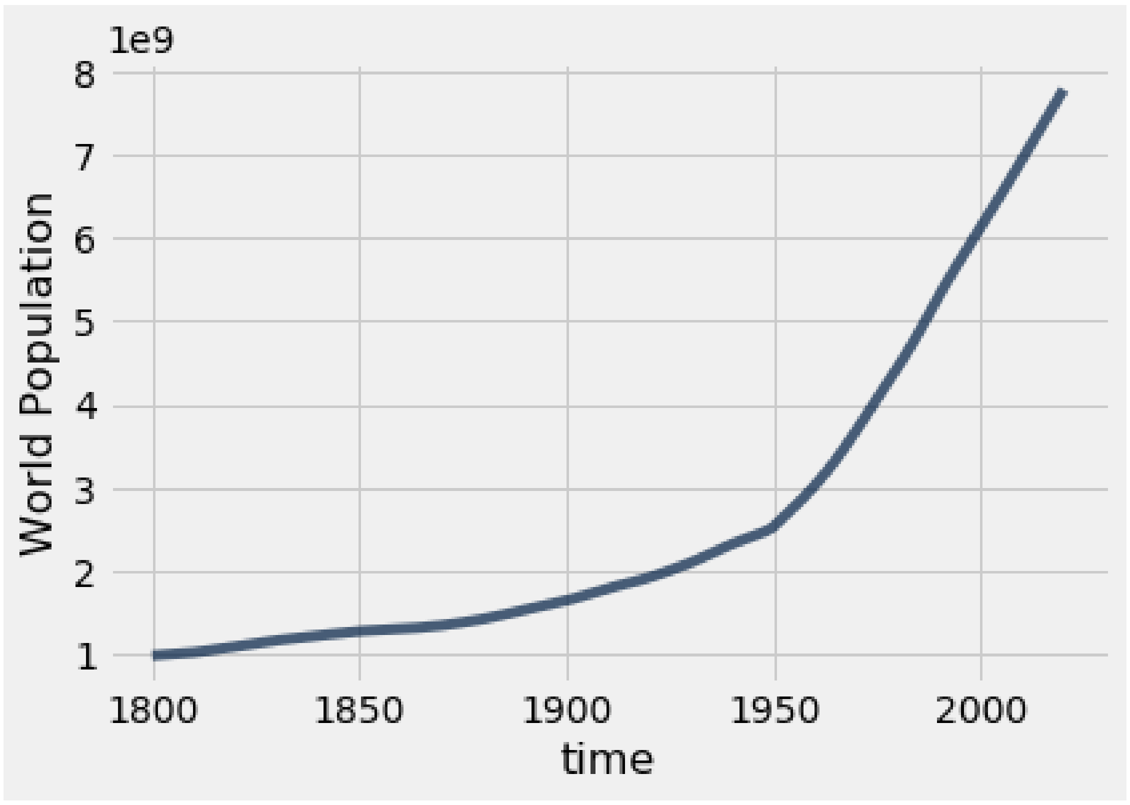
[109]:
```
fertility_statements = 1, 3, 5, 6
```

[110]:
```
ok.grade("q1_9");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 10. Draw a line plot of the world population from 1800 through 2020. The world population is the sum of all the country's populations.

[111]:
```
pop_over_time = population.group("time", sum).drop("count", "geo", "geo sum"
                        ).relabel("population_total sum", "World Population"
                        ).plot("time", "World Population")
```



Question 11. Create a function `stats_for_year` that takes a `year` and returns a table of statistics. The table it returns should have four columns: `geo`, `population_total`, `children_per_woman_total_fertility`, and `child_mortality_under_5_per_1000_born`. Each row should contain one Alpha-3 country code and three statistics: population, fertility rate, and child mortality for that `year` from the `population`, `fertility` and `child_mortality` tables. Only include rows for which all three statistics are available for the country and year.

In addition, restrict the result to country codes that appears in `big_50`, an array of the 50 most populous countries in 2020. This restriction will speed up computations later in the project.

After you write `stats_for_year`, try calling `stats_for_year` on any year between 1960 and 2020. Try to understand the output of stats_for_year.

Hint: The tests for this question are quite comprehensive, so if you pass the tests, your function is probably correct. However, without calling your function yourself and looking at the output, it will be very difficult to understand any problems you have, so try your best to write the function correctly and check that it works before you rely on the `ok` tests to confirm your work.

[112]:
```
# We first create a population table that only includes the
# 50 countries with the largest 2010 populations. We focus on
# these 50 countries only so that plotting later will run faster.
big_50 = population.where('time', are.equal_to(2020)).sort("population_total",␣
↪descending=True).take(np.arange(50)).column('geo')
```

```
population_of_big_50 = population.where('time', are.above(1959)).where('geo',␣
↪are.contained_in(big_50))

def stats_for_year(year):
    """Return a table of the stats for each country that year."""
    p = population_of_big_50.where('time', are.equal_to(year)).drop('time')
    f = fertility.where('time', are.equal_to(year)).drop('time')
    c = child_mortality.where('time', are.equal_to(year)).drop('time')
    geo = population_of_big_50.where("time", are.equal_to(year)).
↪drop("population_total")
    stats_table = geo.join("geo", p).join("geo", f).join("geo", c).drop("time")
    return stats_table

stats_for_year(2010)
```

[112]:
```
geo  | population_total | children_per_woman_total_fertility |
child_mortality_under_5_per_1000_born
afg  | 29185511         | 5.82                               | 87.95
ago  | 23356247         | 6.16                               | 120.49
arg  | 40895751         | 2.37                               | 14.44
bgd  | 147575433        | 2.28                               | 49.1
bra  | 195713637        | 1.81                               | 18.71
can  | 34147566         | 1.64                               | 5.65
chn  | 1368810604       | 1.59                               | 15.76
cod  | 64563853         | 6.47                               | 115.29
col  | 45222699         | 2.01                               | 18.47
deu  | 80827001         | 1.39                               | 4.18
… (40 rows omitted)
```

[113]:
```
ok.grade("q1_11");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 12. Create a table called `pop_by_decade` with two columns called `decade` and `population`. It has a row for each `year` since 1960 that starts a decade. The `population` column contains the total population of all countries included in the result of `stats_for_year(year)` for the first `year` of the decade. For example, 1960 is the first year of the 1960's decade. You should see that these countries contain most of the world's population.

Hint: One approach is to define a function `pop_for_year` that computes this total population, then `apply` it to the `decade` column. The `stats_for_year` function from the previous question may be useful here.

This first test is just a sanity check for your helper function if you choose to use it. You will not lose points for not implementing the function `pop_for_year`.

Note: The cell where you will generate the `pop_by_decade` table is below the cell where you can choose to define the helper function `pop_for_year`. You should define your `pop_by_decade` table in the cell that starts with the table `decades` being defined.

```
[114]: def pop_for_year(year):
           population = sum(stats_for_year(year).column("population_total"))
           return population

       decade = Table().with_columns("decade", np.arange(1960, 2021, 10))
       pop_by_decade = decade.with_column("population", decade.apply(pop_for_year,␣
        ↪"decade"))
       pop_by_decade
```

```
[114]: decade | population
       1960   | 2635123897
       1970   | 3221457416
       1980   | 3890044418
       1990   | 4656339803
       2000   | 5377062169
       2010   | 6064674132
       2020   | 6765161289
```

```
[115]: ok.grade("q1_12_0");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[ooooooooook] 100.0% passed
```

Now that you've defined your helper function (if you've chosen to do so), define the `pop_by_decade` table.

```
[116]: decades = Table().with_column('decade', np.arange(1960, 2021, 10))

       pop_by_decade = pop_by_decade   # we had already defined our helper table␣
        ↪(pop_by_decade) above
       pop_by_decade.set_format(1, NumberFormatter)
```

```
[116]: decade | population
       1960   | 2,635,123,897
       1970   | 3,221,457,416
       1980   | 3,890,044,418
       1990   | 4,656,339,803
       2000   | 5,377,062,169
       2010   | 6,064,674,132
       2020   | 6,765,161,289
```

```
[117]: ok.grade("q1_12");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

The `countries` table describes various characteristics of countries. The `country` column contains the same codes as the `geo` column in each of the other data tables (`population`, `fertility`, and `child_mortality`). The `world_6region` column classifies each country into a region of the world. Run the cell below to inspect the data.

```
[118]: countries = Table.read_table('countries.csv').where('country', are.
        ↪contained_in(population.group('geo').column('geo')))
       countries.select('country', 'name', 'world_6region')
```

```
[118]: country | name                 | world_6region
       afg     | Afghanistan          | south_asia
       ago     | Angola               | sub_saharan_africa
       alb     | Albania              | europe_central_asia
       and     | Andorra              | europe_central_asia
       are     | United Arab Emirates | middle_east_north_africa
       arg     | Argentina            | america
       arm     | Armenia              | europe_central_asia
       atg     | Antigua and Barbuda  | america
       aus     | Australia            | east_asia_pacific
       aut     | Austria              | europe_central_asia
       … (187 rows omitted)
```

Question 13. Create a table called `region_counts` that has two columns, `region` and `count`. It should contain two columns: a region column and a count column that contains the number of countries in each region that appear in the result of `stats_for_year(2020)`. For example, one row would have `south_asia` as its `world_6region` value and an integer as its `count` value: the number of large South Asian countries for which we have population, fertility, and child mortality numbers from 2020.

[119]:
```
region_counts = countries.where(
    "country", are.contained_in(big_50)
    ).group("world_6region").relabel("world_6region", "region")
region_counts
```

[119]:
```
region                     | count
america                    | 8
east_asia_pacific          | 9
europe_central_asia        | 10
middle_east_north_africa   | 7
south_asia                 | 5
sub_saharan_africa         | 11
```

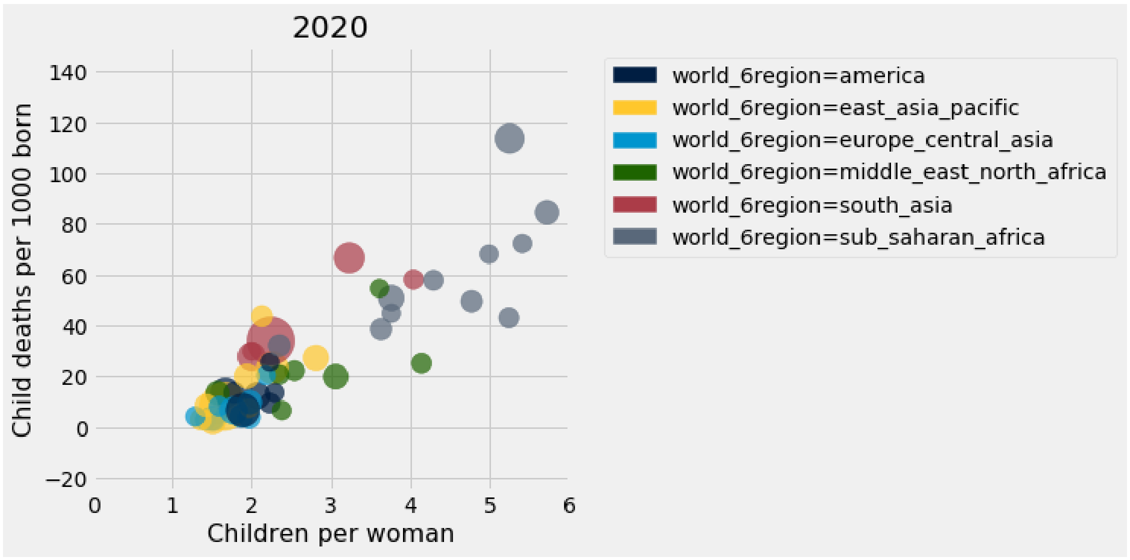
[120]:
```
ok.grade("q1_13");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

The following scatter diagram compares total fertility rate and child mortality rate for each country in 2020. The area of each dot represents the population of the country, and the color represents its region of the world. Run the cell. Do you think you can identify any of the dots?

[121]:
```
from functools import lru_cache as cache

# This cache annotation makes sure that if the same year
# is passed as an argument twice, the work of computing
# the result is only carried out once.
@cache(None)
def stats_relabeled(year):
    """Relabeled and cached version of stats_for_year."""
    return stats_for_year(year).relabel(2, 'Children per woman').relabel(3,␣
 ↪'Child deaths per 1000 born')

def fertility_vs_child_mortality(year):
    """Draw a color scatter diagram comparing child mortality and fertility."""
    with_region = stats_relabeled(year).join('geo', countries.select('country',␣
 ↪'world_6region'), 'country')
    with_region.scatter(2, 3, sizes=1, group=4, s=500)
    plots.xlim(0,6)
```

```
    plots.ylim(-25, 150)
    plots.title(year)

fertility_vs_child_mortality(2020)
```



Question 14. Assign `scatter_statements` to an array of the numbers of each statement below that can be inferred from this scatter diagram for 2020. 1. As a whole, the `europe_central_asia` region had the lowest child mortality rate. 1. The lowest child mortality rate of any country was from an `east_asia_pacific` country. 1. Most countries had a fertility rate above 5. 1. There was an association between child mortality and fertility. 1. The two largest countries by population also had the two highest child mortality rate.

```
[122]: scatter_statements = 2,4
```

```
[123]: ok.grade("q1_14");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

The result of the cell below is interactive. Drag the slider to the right to see how countries have changed over time. You'll find that the great divide between so-called "Western" and "developing" countries that existed in the 1960's has nearly disappeared. This shift in fertility rates is the reason that the global population is expected to grow more slowly in the 21st century than it did in the

19th and 20th centuries.

Note: Don't worry if a red warning pops up when running the cell below. You'll still be able to run the cell!

[124]:
```
import ipywidgets as widgets

# This part takes a few minutes to run because it
# computes 55 tables in advance: one for each year.
Table().with_column('Year', np.arange(1960, 2016)).apply(stats_relabeled,␣
 ↪'Year')

_ = widgets.interact(fertility_vs_child_mortality,
                     year=widgets.IntSlider(min=1960, max=2015, value=1960))
```

```
/opt/conda/lib/python3.8/site-packages/datascience/tables.py:193: FutureWarning:
Implicit column method lookup is deprecated.
  warnings.warn("Implicit column method lookup is deprecated.", FutureWarning)
```

```
interactive(children=(IntSlider(value=1960, description='year', max=2015, min=1960), Output())
```

Now is a great time to take a break and watch the same data presented by Hans Rosling in a 2010 TEDx talk with smoother animation and witty commentary.

0.3 2. Global Poverty

In 1800, 85% of the world's 1 billion people lived in *extreme poverty*, defined by the United Nations as "a condition characterized by severe deprivation of basic human needs, including food, safe drinking water, sanitation facilities, health, shelter, education and information." A common measure of extreme poverty is a person living on less than $1.25 per day.

In 2018, the proportion of people living in extreme poverty was estimated to be 8%. Although the world rate of extreme poverty has declined consistently for hundreds of years, the number of people living in extreme poverty is still over 600 million. The United Nations recently adopted an ambitious goal: "By 2030, eradicate extreme poverty for all people everywhere." In this section, we will examine extreme poverty trends around the world.

First, load the population and poverty rate by country and year and the country descriptions. While the `population` table has values for every recent year for many countries, the `poverty` table only includes certain years for each country in which a measurement of the rate of extreme poverty was available.

[125]:
```
population = Table.read_table('population.csv')
countries = Table.read_table('countries.csv').where('country', are.
 ↪contained_in(population.group('geo').column('geo')))
poverty = Table.read_table('poverty.csv')
population
```

```
[125]: geo  | time | population_total
       afg  | 1800 | 3280000
       afg  | 1801 | 3280000
       afg  | 1802 | 3280000
       afg  | 1803 | 3280000
       afg  | 1804 | 3280000
       afg  | 1805 | 3280000
       afg  | 1806 | 3280000
       afg  | 1807 | 3280000
       afg  | 1808 | 3280000
       afg  | 1809 | 3280000
       … (59287 rows omitted)
```

Question 1. Assign `latest_poverty` to a three-column table with one row for each country that appears in the `poverty` table. The first column should contain the 3-letter code for the country. The second column should contain the most recent year for which an extreme poverty rate is available for the country. The third column should contain the poverty rate in that year. **Do not change the last line, so that the labels of your table are set correctly.**

Hint: think about how `group` works: it does a sequential search of the table (from top to bottom) and collects values in the array in the order in which they appear, and then applies a function to that array. The `first` function may be helpful, but you are not required to use it.

```
[126]: def first(values):
           return values.item(0)
       # population.where("time", 2010).join("geo", poverty, "geo") #
       latest_poverty = poverty.sort("time", descending=True).group("geo", first)
       latest_poverty = latest_poverty.relabeled(0, 'geo').relabeled(1, 'time').
        ↪relabeled(2, 'poverty_percent') # You should *not* change this line.
       latest_poverty
```

```
[126]: geo  | time | poverty_percent
       ago  | 2009 | 43.37
       alb  | 2012 | 0.46
       arg  | 2011 | 1.41
       arm  | 2012 | 1.75
       aus  | 2003 | 1.36
       aut  | 2004 | 0.34
       aze  | 2008 | 0.31
       bdi  | 2006 | 81.32
       bel  | 2000 | 0.5
       ben  | 2012 | 51.61
       … (135 rows omitted)
```

```
[127]: ok.grade("q2_1");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests
```

```
---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 2. Using both `latest_poverty` and `population`, create a four-column table called `recent_poverty_total` with one row for each country in `latest_poverty`. The four columns should have the following labels and contents: 1. `geo` contains the 3-letter country code, 1. `poverty_percent` contains the most recent poverty percent, 1. `population_total` contains the population of the country in 2010, 1. `poverty_total` contains the number of people in poverty **rounded to the nearest integer**, based on the 2010 population and most recent poverty rate.

[128]:
```
poverty_and_pop = population.where("time", 2010).join("geo", latest_poverty).
 ↪drop("time", "time_2")
recent_poverty_total = poverty_and_pop.with_columns(
    "poverty_total",
                    np.round(
                    (poverty_and_pop.column("population_total") *
                    (poverty_and_pop.column("poverty_percent")/100))
    )).select("geo", "poverty_percent", "population_total", "poverty_total")
recent_poverty_total
```

[128]:
```
geo  | poverty_percent | population_total | poverty_total
ago  | 43.37           | 23356247         | 1.01296e+07
alb  | 0.46            | 2948029          | 13561
arg  | 1.41            | 40895751         | 576630
arm  | 1.75            | 2877314          | 50353
aus  | 1.36            | 22154687         | 301304
aut  | 0.34            | 8409945          | 28594
aze  | 0.31            | 9032465          | 28001
bdi  | 81.32           | 8675606          | 7.055e+06
bel  | 0.5             | 10938735         | 54694
ben  | 51.61           | 9199254          | 4.74774e+06
… (135 rows omitted)
```

[129]:
```
ok.grade("q2_2");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 3. Assign the name `poverty_percent` to the known percentage of the world's 2010 population that were living in extreme poverty. Assume that the `poverty_total` numbers in the `recent_poverty_total` table describe **all** people in 2010 living in extreme poverty. You should find a number that is above the 2018 global estimate of 8%, since many country-specific poverty rates are older than 2018.

Hint: The sum of the `population_total` column in the `recent_poverty_total` table is not the world population, because only a subset of the world's countries are included in the `recent_poverty_total` table (only some countries have known poverty rates). Use the `population` table to compute the world's 2010 total population..

[130]:
```
poverty_percent = (sum(recent_poverty_total.column("poverty_total")) * 100) /␣
 ↪sum(population.where("time", are.equal_to(2010)).column("population_total"))
poverty_percent
```

[130]:
```
14.248865303997139
```

[131]:
```
ok.grade("q2_3");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

The `countries` table includes not only the name and region of countries, but also their positions on the globe.

[132]:
```
countries.select('country', 'name', 'world_4region', 'latitude', 'longitude')
```

[132]:
```
country | name                 | world_4region | latitude | longitude
afg     | Afghanistan          | asia          | 33       | 66
ago     | Angola               | africa        | -12.5    | 18.5
alb     | Albania              | europe        | 41       | 20
and     | Andorra              | europe        | 42.5078  | 1.52109
are     | United Arab Emirates | asia          | 23.75    | 54.5
arg     | Argentina            | americas      | -34      | -64
arm     | Armenia              | europe        | 40.25    | 45
atg     | Antigua and Barbuda  | americas      | 17.05    | -61.8
aus     | Australia            | asia          | -25      | 135
aut     | Austria              | europe        | 47.3333  | 13.3333
… (187 rows omitted)
```

```
[133]: countries
```

```
[133]: country | g77_and_oecd_countries | income_groups       | is--country |
       iso3166_1_alpha2 | iso3166_1_alpha3 | iso3166_1_numeric | iso3166_2 | landlocked
       | latitude | longitude | main_religion_2008 | name                 | un_state |
       unicef_region | unicode_region_subtag | world_4region | world_6region
       afg     | g77                    | low_income          | True        | AF
       | AFG              | 4                 | nan       | landlocked | 33       | 66
       | muslim             | Afghanistan          | True     | sa            | AF
       | asia          | south_asia
       ago     | g77                    | upper_middle_income | True        | AO
       | AGO              | 24                | nan       | coastline  | -12.5    |
       18.5      | christian          | Angola               | True     | ssa
       | AO                    | africa        | sub_saharan_africa
       alb     | others                 | upper_middle_income | True        | AL
       | ALB              | 8                 | nan       | coastline  | 41       | 20
       | muslim             | Albania              | True     | eca           | AL
       | europe        | europe_central_asia
       and     | others                 | high_income         | True        | AD
       | AND              | 20                | nan       | landlocked | 42.5078  |
       1.52109   | christian          | Andorra              | True     | eca
       | AD                    | europe        | europe_central_asia
       are     | g77                    | high_income         | True        | AE
       | ARE              | 784               | nan       | coastline  | 23.75    |
       54.5      | muslim             | United Arab Emirates | True     | mena
       | AE                    | asia          | middle_east_north_africa
       arg     | g77                    | upper_middle_income | True        | AR
       | ARG              | 32                | nan       | coastline  | -34      | -64
       | christian          | Argentina            | True     | lac           | AR
       | americas      | america
       arm     | others                 | lower_middle_income | True        | AM
       | ARM              | 51                | nan       | landlocked | 40.25    | 45
       | christian          | Armenia              | True     | eca           | AM
       | europe        | europe_central_asia
       atg     | g77                    | high_income         | True        | AG
       | ATG              | 28                | nan       | coastline  | 17.05    |
       -61.8     | christian          | Antigua and Barbuda  | True     | lac
       | AG                    | americas      | america
       aus     | oecd                   | high_income         | True        | AU
       | AUS              | 36                | nan       | coastline  | -25      | 135
       | christian          | Australia            | True     | eap           | AU
       | asia          | east_asia_pacific
       aut     | oecd                   | high_income         | True        | AT
       | AUT              | 40                | nan       | landlocked | 47.3333  |
       13.3333   | christian          | Austria              | True     | eca
       | AT                    | europe        | europe_central_asia
       … (187 rows omitted)
```

Question 4. Using both `countries` and `recent_poverty_total`, create a five-column table called `poverty_map` with one row for every country in `recent_poverty_total`. The five columns should have the following labels and contents: 1. `latitude` contains the country's latitude, 1. `longitude` contains the country's longitude, 1. `name` contains the country's name, 1. `region` contains the country's region from the `world_4region` column of `countries`, 1. `poverty_total` contains the country's poverty total.

[134]:
```
poverty_map = recent_poverty_total.join("geo", countries, "country"
            ).select("latitude", "longitude",
                     "name", "world_4region",
                     "poverty_total"
            ).relabel("world_4region", "region")
poverty_map
```

[134]:
```
latitude | longitude | name       | region   | poverty_total
-12.5    | 18.5      | Angola     | africa   | 1.01296e+07
41       | 20        | Albania    | europe   | 13561
-34      | -64       | Argentina  | americas | 576630
40.25    | 45        | Armenia    | europe   | 50353
-25      | 135       | Australia  | asia     | 301304
47.3333  | 13.3333   | Austria    | europe   | 28594
40.5     | 47.5      | Azerbaijan | europe   | 28001
-3.5     | 30        | Burundi    | africa   | 7.055e+06
50.75    | 4.5       | Belgium    | europe   | 54694
9.5      | 2.25      | Benin      | africa   | 4.74774e+06
… (135 rows omitted)
```

[135]:
```
ok.grade("q2_4");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

Run the cell below to draw a map of the world in which the areas of circles represent the number of people living in extreme poverty. Double-click on the map to zoom in.

[136]:
```
# It may take a few seconds to generate this map.
colors = {'africa': 'blue', 'europe': 'black', 'asia': 'red', 'americas':␣
 ↪'green'}
scaled = poverty_map.with_columns(
    'poverty_total', 1e-4 * poverty_map.column('poverty_total'),
    'region', poverty_map.apply(colors.get, 'region')
```

```
)
Circle.map_table(scaled)
```

[136]:
```
<datascience.maps.Map at 0x7f4a25c0b7c0>
```

Although people live in extreme poverty throughout the world (with more than 5 million in the United States), the largest numbers are in Asia and Africa.

Question 5. Assign `largest` to a two-column table with the `name` (not the 3-letter code) and `poverty_total` of the 10 countries with the largest number of people living in extreme poverty.

[137]:
```
largest = countries.select("name"
                  ).join("name", poverty_map, "name"
                  ).sort("poverty_total", descending = True
                  ).select("name", "poverty_total"
                  ).take(np.arange(0, 10))
largest
largest.set_format('poverty_total', NumberFormatter)
```

[137]:
```
name                | poverty_total
India               | 291,660,639.00
Nigeria             | 98,319,537.00
China               | 85,687,544.00
Bangladesh          | 63,826,375.00
Congo, Dem. Rep.    | 56,635,412.00
Indonesia           | 39,177,145.00
Ethiopia            | 32,242,742.00
Pakistan            | 22,858,700.00
Tanzania            | 19,281,872.00
Madagascar          | 18,543,643.00
```

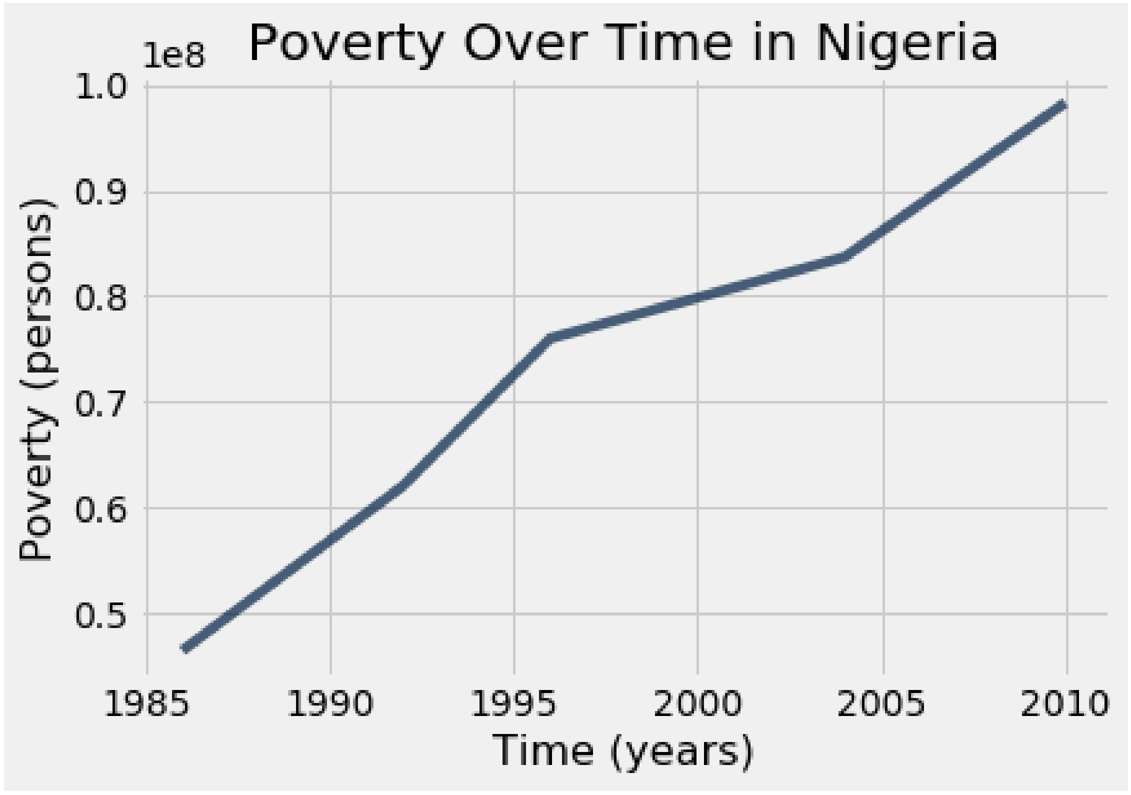
[138]:
```
ok.grade("q2_5");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 6. Write a function called `poverty_timeline` that takes **the name of a country** as its argument. It should draw a line plot of the number of people living in poverty in that country with time on the horizontal axis. The line plot should have a point for each row in the `poverty` table for that country. To compute the population living in poverty from a poverty percentage, multiply by the population of the country **in that year**.

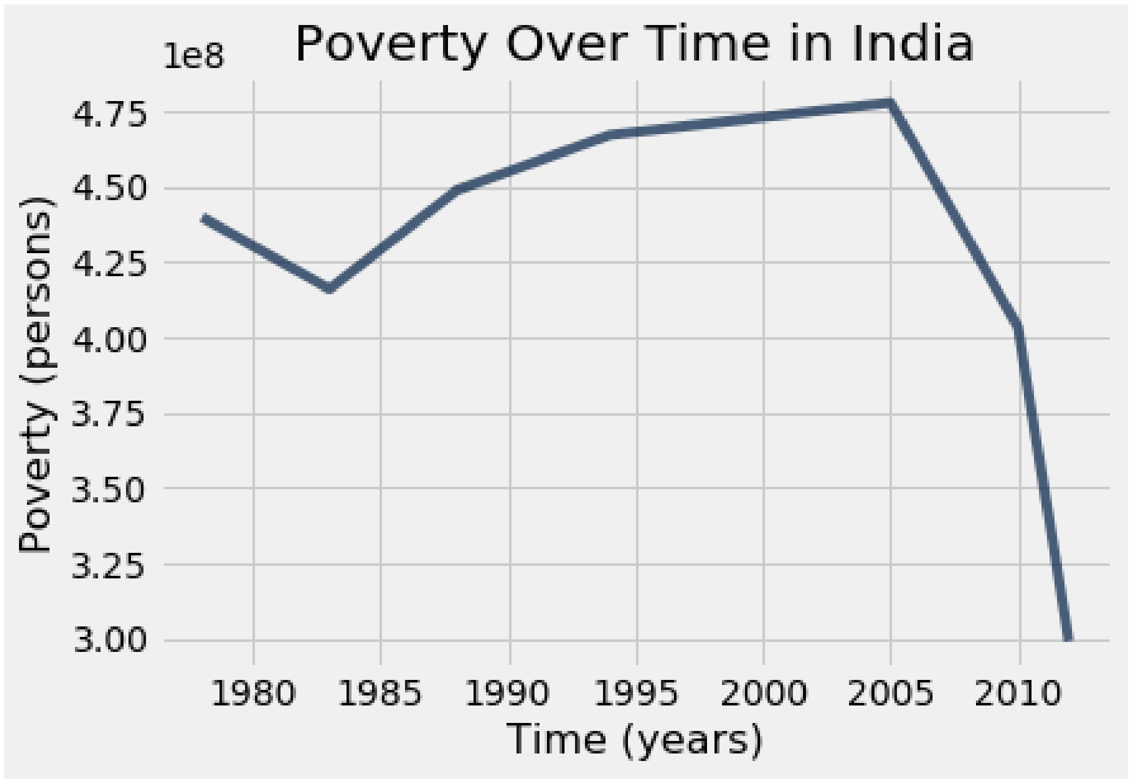
Hint: This question is long. Feel free to create cells and experiment.

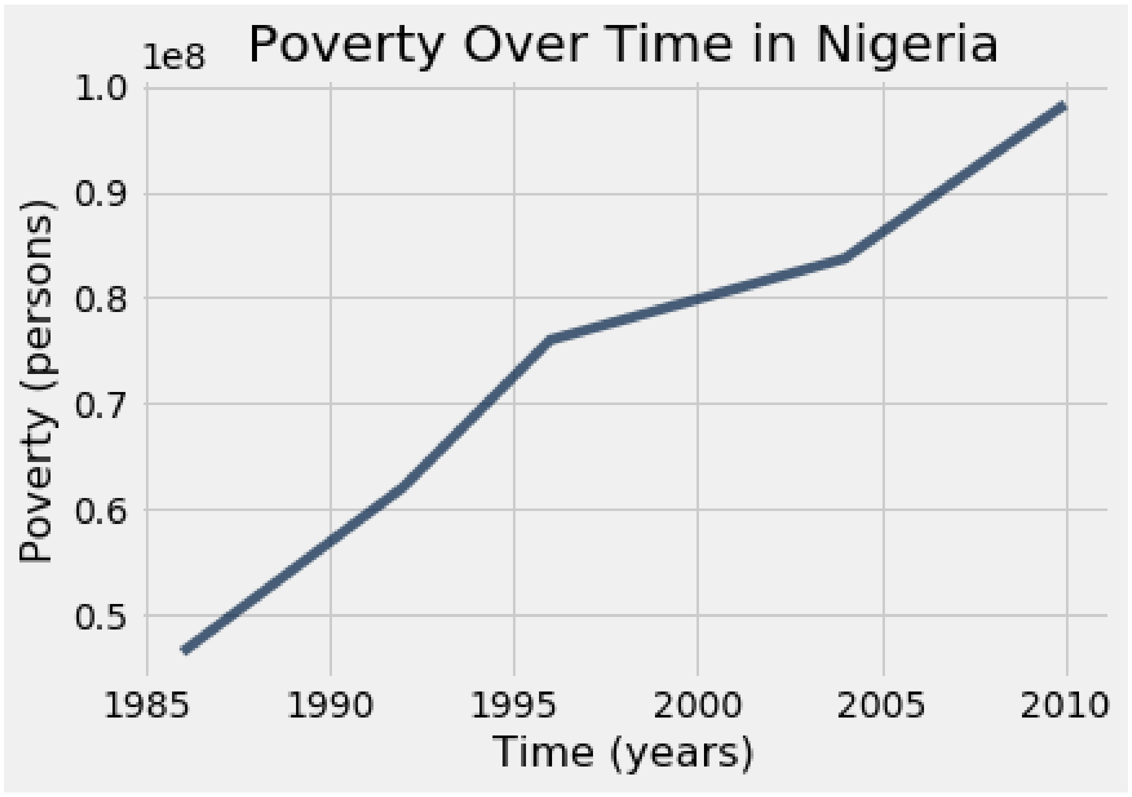
```
[139]: import matplotlib.pyplot as plt

def poverty_timeline(country):
    '''Draw a timeline of people living in extreme poverty in a country.'''
    pop_pov_percent_all = poverty.join(["geo", "time"], population
                          ).join("geo", countries, "country"
                          ).select("name", "time", "population_total",␣
 ↪"extreme_poverty_percent_people_below_125_a_day")
    poverty_all = pop_pov_percent_all.with_columns(
    "poverty_total", np.round(
                        (pop_pov_percent_all.column("population_total") *
                        (pop_pov_percent_all.
 ↪column("extreme_poverty_percent_people_below_125_a_day")/100)))
                                                       )
    geo = poverty_all.where("name", are.equal_to(country))
    plot_table = geo.select("time", "poverty_total")
    plotted = plot_table.plot("time", "poverty_total")
    plt.title('Poverty Over Time in ' + str(country))
    plt.xlabel("Time (years)")
    plt.ylabel("Poverty (persons)")
    # This solution will take multiple lines of code. Use as many as you need
    return plotted
poverty_timeline('Nigeria')
```

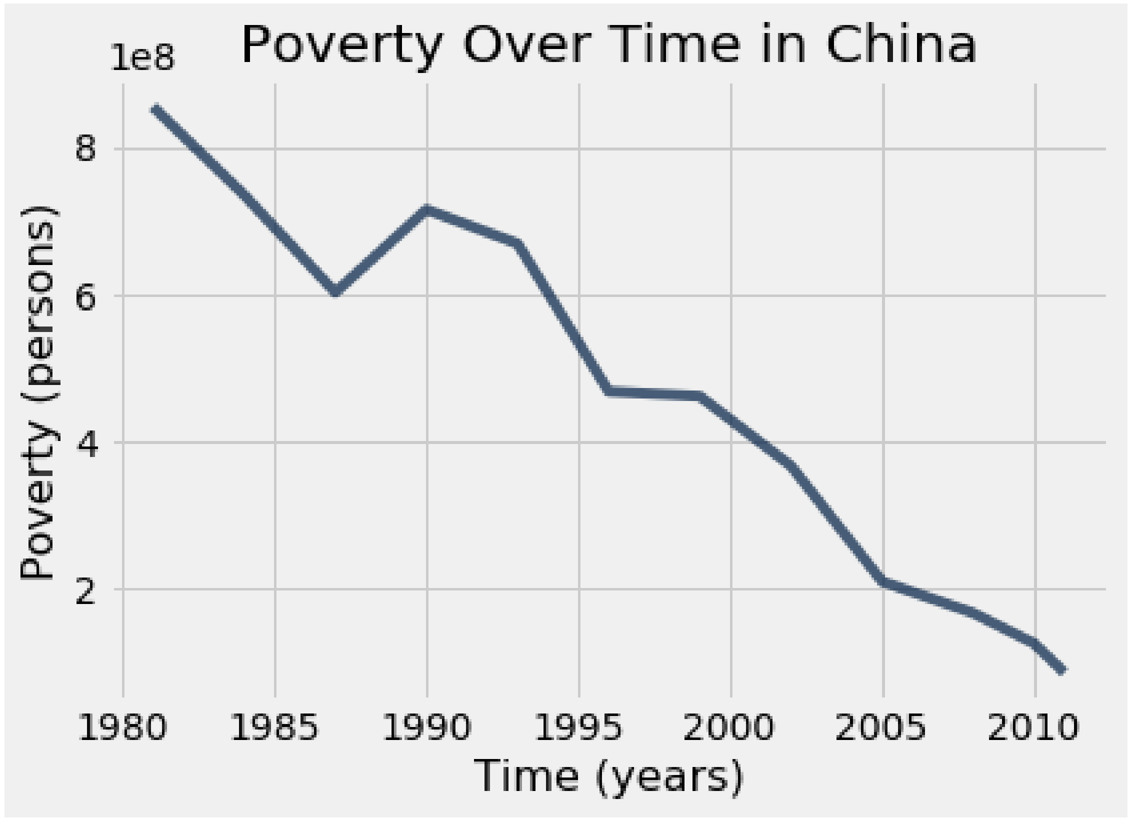


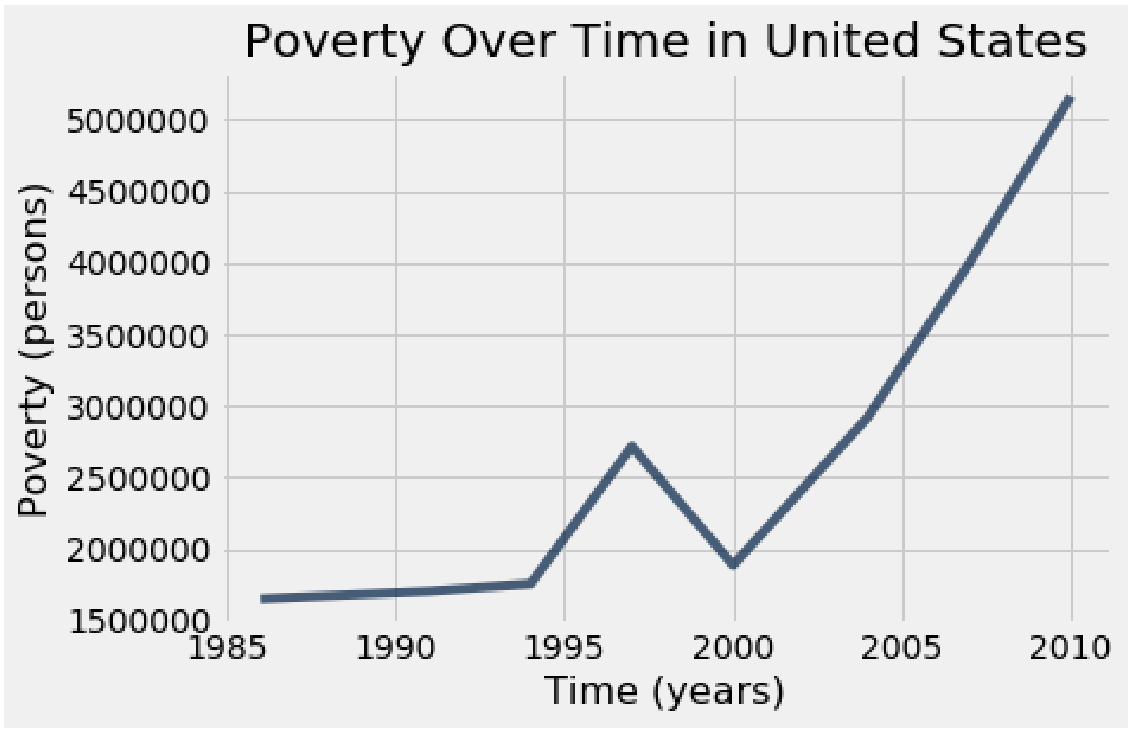
Finally, draw the timelines below to see how the world is changing. You can check your work by comparing your graphs to the ones on gapminder.org.

```
[140]: poverty_timeline('India')
       poverty_timeline('Nigeria')
       poverty_timeline('China')
       poverty_timeline('United States')
```




Poverty Over Time in Nigeria
1e8
1.0
0.9
0.8
0.7
0.6
0.5
Poverty (persons)
1985
1990
1995
2000
2005
2010
Time (years)


Poverty Over Time in China
1e8
8
6
4
2
Poverty (persons)
1980
1985
1990
1995
2000
2005
2010
Time (years)



Although the number of people living in extreme poverty has been increasing in Nigeria and the United States, the massive decreases in China and India have shaped the overall trend that extreme poverty is decreasing worldwide, both in percentage and in absolute number.

To learn more, watch Hans Rosling in a 2015 film about the UN goal of eradicating extreme poverty from the world.

Below, we've also added an interactive dropdown menu for you to visualize `poverty_timeline` graphs for other countries. Note that each dropdown menu selection may take a few seconds to run.

[141]:
```
# Just run this cell

all_countries = poverty_map.column('name')
_ = widgets.interact(poverty_timeline, country=list(all_countries))
```

```
interactive(children=(Dropdown(description='country', options=('Angola', 'Albania', 'Argentina
```

You're finished! Congratulations on mastering data visualization and table manipulation. Time to submit.

0.4 3. Submission

Once you're finished, select "Save and Checkpoint" in the File menu and then execute the `submit` cell below. The result will contain a link that you can use to check that your assignment has been submitted successfully. If you submit more than once before the deadline, we will only grade your final submission. If you mistakenly submit the wrong one, you can head to okpy.org and flag the correct version. To do so, go to the website, click on this assignment, and find the version you would like to have graded. There should be an option to flag that submission for grading!

```
[ ]: _ = ok.submit()
```

```
[143]: # For your convenience, you can run this cell to run all the tests at once!
import os
print("Running all tests...")
_ = [ok.grade(q[:-3]) for q in os.listdir("tests") if q.startswith('q') and␣
↪len(q) <= 10]
print("Finished running all tests.")
```

```
Running all tests…
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests
```

```
---------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
```

```
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooook] 100.0% passed

Finished running all tests.
```