Refugees

June 26, 2018

1 Pandas

- https://pandas.pydata.org
- very high-level data containers with corresponding functionality
- many useful tools to work with time-series (look at `Series.rolling`)
- many SQL-like data operations (group, join, merge)
- Interface to a large variety of file formats (see `pd.read_[...]` functions)
- additional package with data-interface/API to many data repositories (https://pandas-datareader.readthedocs.io/en/latest/remote_data.html)

```
In [1]: import pandas as pd
```

1.1 Basic Data Structures

1.1.1 Series

One-dimensional ndarray with axis labels (called index).

Series can be created like an array

```
In [2]: pd.Series([11,13,17,19,23])
```

```
Out[2]: 0    11
        1    13
        2    17
        3    19
        4    23
        dtype: int64
```

or, if you want a special index

```
In [3]: series = pd.Series([11,13,17,19,23], index=['a', 'b', 'c', 'd', 'e'])
        print(series)
```

```
a    11
b    13
c    17
d    19
e    23
dtype: int64
```

to get the content back you can use

```
In [4]: series.index

Out[4]: Index(['a', 'b', 'c', 'd', 'e'], dtype='object')

In [5]: series.values

Out[5]: array([11, 13, 17, 19, 23])
```

but the power of pandas lies in all the other attributes

```
In [6]: #series. [TAB]
```

1.1.2 DataFrame

The primary pandas data structure.

Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes. (index: row labels, columns: column labels) Can be thought of as a dict-like container for Series objects.

The easiest way to create a DataFrame is to read it from an input file (see later)

In addition there are many ways to create DataFrames manually. Most straight forward probably is to use a dict of iterables. (Series, Lists, Arrays). Pandas tries to choose sensible indexes.

```
In [7]: pd.DataFrame({"primes": series, "fibo": [1,1,2,3,5]})

Out[7]:    fibo  primes
        a     1      11
        b     1      13
        c     2      17
        d     3      19
        e     5      23
```

2 Refugee Example

We now want to use pandas to work with data from the World Bank. My goal is to create a plot showing the burden refugees put on different countries. For this we will plot the fraction of refugee in a give countries population versus that countries GDP.

I downloaded and extracted the following data-sets from the Worldbank website manually: * Refugee population by country or territory of asylum: https://data.worldbank.org/indicator/SM.POP.REFG * Population, total: https://data.worldbank.org/indicator/SP.POP.TOTL * GDP per capita (current US$): https://data.worldbank.org/indicator/NY.GDP.PCAP.CD

```
In [8]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

2.1 Loading and Accessing Data

loading a data file with pandas is trivial

```
In [9]: refugees = pd.read_csv("data/refugee-population.csv", skiprows=4)
```

```
In [10]: refugees.head()
```

```
Out[10]:   Country Name Country Code  \
         0        Aruba          ABW
         1  Afghanistan          AFG
         2       Angola          AGO
         3      Albania          ALB
         4      Andorra          AND

                                               Indicator Name Indicator Code  1960  \
         0  Refugee population by country or territory of ...    SM.POP.REFG   NaN
         1  Refugee population by country or territory of ...    SM.POP.REFG   NaN
         2  Refugee population by country or territory of ...    SM.POP.REFG   NaN
         3  Refugee population by country or territory of ...    SM.POP.REFG   NaN
         4  Refugee population by country or territory of ...    SM.POP.REFG   NaN

            1961  1962  1963  1964  1965     ...          2009      2010      2011  \
         0   NaN   NaN   NaN   NaN   NaN     ...           NaN       NaN       NaN
         1   NaN   NaN   NaN   NaN   NaN     ...          37.0    6434.0    3009.0
         2   NaN   NaN   NaN   NaN   NaN     ...       14734.0   15155.0   16223.0
         3   NaN   NaN   NaN   NaN   NaN     ...          70.0      76.0      82.0
         4   NaN   NaN   NaN   NaN   NaN     ...           NaN       NaN       NaN

               2012     2013      2014      2015     2016  2017  Unnamed: 62
         0      NaN      1.0       NaN       2.0      NaN   NaN          NaN
         1  16187.0  16863.0  300423.0  257554.0  59770.0   NaN          NaN
         2  23413.0  23783.0   15474.0   15555.0  15537.0   NaN          NaN
         3     86.0     93.0     104.0     104.0    111.0   NaN          NaN
         4      NaN      NaN       NaN       NaN      NaN   NaN          NaN

         [5 rows x 63 columns]
```

As you can see pandas choose the right column labels and numbered the rows continously. We can easily change the row labels (the index) to one of the columns.

```
In [11]: refugees.set_index(["Country Name"], inplace=True)
```

```
In [12]: refugees.head()
```

```
Out[12]:              Country Code                                     Indicator Name  \
         Country Name
         Aruba                 ABW  Refugee population by country or territory of ...
         Afghanistan           AFG  Refugee population by country or territory of ...
```

```
Angola                AGO  Refugee population by country or territory of ...
Albania               ALB  Refugee population by country or territory of ...
Andorra               AND  Refugee population by country or territory of ...

             Indicator Code  1960  1961  1962  1963  1964  1965  1966  \
Country Name
Aruba           SM.POP.REFG   NaN   NaN   NaN   NaN   NaN   NaN   NaN
Afghanistan     SM.POP.REFG   NaN   NaN   NaN   NaN   NaN   NaN   NaN
Angola          SM.POP.REFG   NaN   NaN   NaN   NaN   NaN   NaN   NaN
Albania         SM.POP.REFG   NaN   NaN   NaN   NaN   NaN   NaN   NaN
Andorra         SM.POP.REFG   NaN   NaN   NaN   NaN   NaN   NaN   NaN

                 ...           2009     2010     2011     2012     2013  \
Country Name     ...
Aruba            ...            NaN      NaN      NaN      NaN      1.0
Afghanistan      ...           37.0   6434.0   3009.0  16187.0  16863.0
Angola           ...        14734.0  15155.0  16223.0  23413.0  23783.0
Albania          ...           70.0     76.0     82.0     86.0     93.0
Andorra          ...            NaN      NaN      NaN      NaN      NaN

                  2014      2015     2016  2017  Unnamed: 62
Country Name
Aruba              NaN       2.0      NaN   NaN          NaN
Afghanistan   300423.0  257554.0  59770.0   NaN          NaN
Angola         15474.0   15555.0  15537.0   NaN          NaN
Albania          104.0     104.0    111.0   NaN          NaN
Andorra            NaN       NaN      NaN   NaN          NaN

[5 rows x 62 columns]
```

Now it's easy to select rows or columns

```
In [13]: refugees.loc[["Switzerland","Germany"]]
```

```
Out[13]:              Country Code                                      Indicator Name  \
         Country Name
         Switzerland           CHE  Refugee population by country or territory of ...
         Germany               DEU  Refugee population by country or territory of ...

                      Indicator Code  1960  1961  1962  1963  1964  1965  1966  \
         Country Name
         Switzerland     SM.POP.REFG   NaN   NaN   NaN   NaN   NaN   NaN   NaN
         Germany         SM.POP.REFG   NaN   NaN   NaN   NaN   NaN   NaN   NaN

                          ...            2009      2010      2011      2012      2013  \
         Country Name     ...
         Switzerland      ...         46203.0   48813.0   50416.0   50747.0   52464.0
         Germany          ...        593799.0  594269.0  571684.0  589737.0  187567.0
```

```
                    2014      2015      2016  2017  Unnamed: 62
Country Name
Switzerland      62620.0   73336.0   82608.0   NaN          NaN
Germany         216973.0  316115.0  669408.0   NaN          NaN

[2 rows x 62 columns]
```

```
In [14]: refugees[["1990","2000"]].head()
```

```
Out[14]:                    1990     2000
         Country Name
         Aruba               NaN      NaN
         Afghanistan        50.0      NaN
         Angola          11557.0  12086.0
         Albania             NaN    523.0
         Andorra             NaN      NaN
```

```
In [15]: refugees.get(["1990","2000"]).head()
```

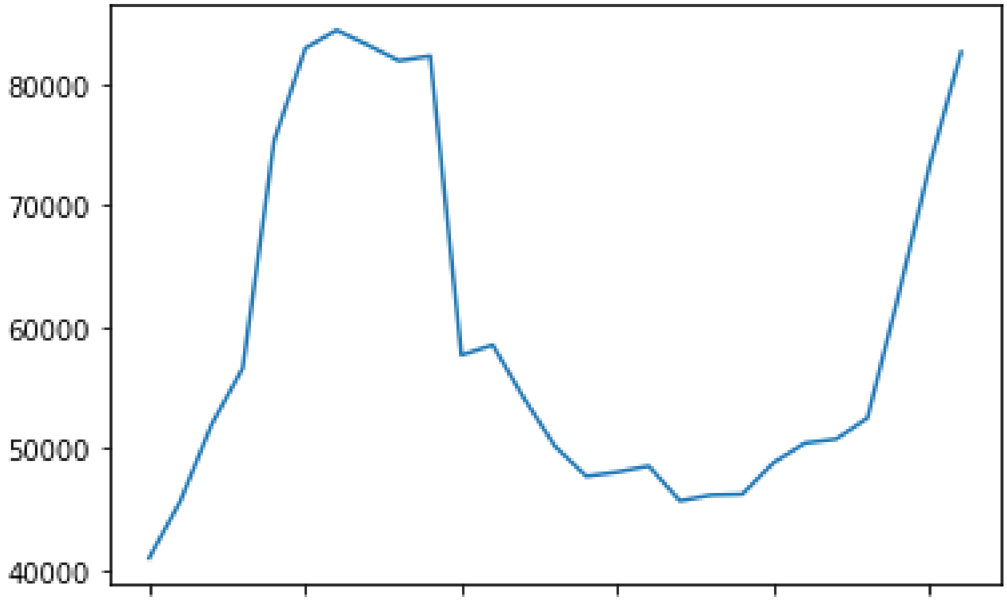
```
Out[15]:                    1990     2000
         Country Name
         Aruba               NaN      NaN
         Afghanistan        50.0      NaN
         Angola          11557.0  12086.0
         Albania             NaN    523.0
         Andorra             NaN      NaN
```

2.2 Working with a Single Country

With this we now choose the data for one country, remove all missing values and then create a plot:

```
In [16]: che = refugees.loc["Switzerland"][["{0}".format(year) for year in range(1990,2018)]]
```

```
In [17]: che.dropna().plot()
         plt.show()
```



Usually it is easier to work with real datetime objects instead of strings. So we convert the index to datetime

```
In [18]: che.index.values
```

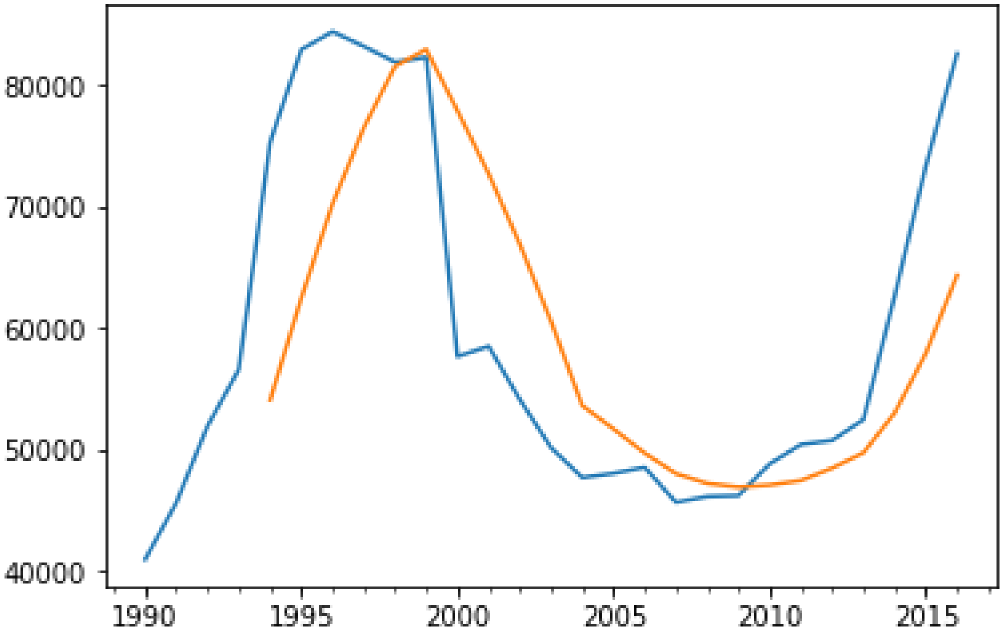
```
Out[18]: array(['1990', '1991', '1992', '1993', '1994', '1995', '1996', '1997',
                '1998', '1999', '2000', '2001', '2002', '2003', '2004', '2005',
                '2006', '2007', '2008', '2009', '2010', '2011', '2012', '2013',
                '2014', '2015', '2016', '2017'], dtype=object)
```

```
In [19]: che.index = pd.to_datetime(che.index, format="%Y")
         print(che.index)
```

```
DatetimeIndex(['1990-01-01', '1991-01-01', '1992-01-01', '1993-01-01',
               '1994-01-01', '1995-01-01', '1996-01-01', '1997-01-01',
               '1998-01-01', '1999-01-01', '2000-01-01', '2001-01-01',
               '2002-01-01', '2003-01-01', '2004-01-01', '2005-01-01',
               '2006-01-01', '2007-01-01', '2008-01-01', '2009-01-01',
               '2010-01-01', '2011-01-01', '2012-01-01', '2013-01-01',
               '2014-01-01', '2015-01-01', '2016-01-01', '2017-01-01'],
              dtype='datetime64[ns]', freq=None)
```

As mentioned in the introduction, `pandas` offers a very usefull `rolling` method

```
In [20]: che.plot()
         che.rolling(center=False,window=5).mean().plot()
         plt.show()
```



2.3 Removing Unwanted Data

We now want to create a scatter plot with refugees divided by gdp vs. gdp-per-captita. For each data set we will use the mean of the last 7 years.

Some of the rows and columns in the World-Bank Files are of no interest for this. We can remove these easily.

2.3.1 Excluding Non-Countries

The World-Bank provides meta-data for each country, where we can identify rows with non-countries (e.g. regional aggregates)

```
In [21]: !head data/metadata-countries_population.csv
```

```
"ARG","Latin America & Caribbean","Upper middle income","National Institute of Statistics and C
```

We load this file and extract the two relevant columns

```
In [22]: meta = pd.read_csv("data/metadata-countries_population.csv")
```

```
In [23]: meta.columns
```

```
Out[23]: Index(['Country Code', 'Region', 'IncomeGroup', 'SpecialNotes', 'TableName',
                'Unnamed: 5'],
               dtype='object')
```

```
In [24]: meta = meta[['Country Code', 'Region']]
```

```
In [25]: meta.head()
```

```
Out[25]:   Country Code                      Region
         0          ABW  Latin America & Caribbean
         1          AFG                 South Asia
         2          AGO         Sub-Saharan Africa
         3          ALB      Europe & Central Asia
         4          AND      Europe & Central Asia
```

```
In [26]: meta.set_index("Country Code", inplace=True)
```

From this we create a list of non-countries

```
In [27]: non_countries = meta.loc[meta.Region.isnull()].index
         print(non_countries)
```

```
Index(['ARB', 'CEB', 'CSS', 'EAP', 'EAR', 'EAS', 'ECA', 'ECS', 'EMU', 'EUU',
       'FCS', 'HIC', 'HPC', 'IBD', 'IBT', 'IDA', 'IDB', 'IDX', 'LAC', 'LCN',
       'LDC', 'LIC', 'LMC', 'LMY', 'LTE', 'MEA', 'MIC', 'MNA', 'NAC', 'OED',
       'OSS', 'PRE', 'PSS', 'PST', 'SAS', 'SSA', 'SSF', 'SST', 'TEA', 'TEC',
       'TLA', 'TMN', 'TSA', 'TSS', 'UMC', 'WLD'],
      dtype='object', name='Country Code')
```

and finally exclude the relevant rows

```
In [28]: refugees = refugees[~refugees["Country Code"].isin(non_countries)]
```

2.3.2 Excluding Columns

The data contains a few rows with unneeded text

```
In [29]: refugees.columns
```

```
Out[29]: Index(['Country Code', 'Indicator Name', 'Indicator Code', '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', '2017', 'Unnamed: 62'],
               dtype='object')
```

In addition, the 2017 column is empty

```
In [30]: np.any(refugees["2017"].notnull())
```

```
Out[30]: False
```

so we can create a list of all interesting columns

```
In [31]: useful_cols = ['Country Code']
         for year in range(2010,2017):
             useful_cols.append("{0}".format(year))
```

```
In [32]: useful_cols
```

```
Out[32]: ['Country Code', '2010', '2011', '2012', '2013', '2014', '2015', '2016']
```

with this, we:

- select the reduced datase
- switch the index to Country Code
- calculate the mean for each country

```
In [33]: refugees = refugees[useful_cols]
```

```
In [34]: refugees = refugees.set_index("Country Code")
```

```
In [35]: refugee_means = refugees.mean(axis=1)
```

2.4 Loading Additional Files

Of course we could execute these commands again manually for the two remaining data-files. However, the proper way to solve this is to create a function for this. Especially since all files have the exact same structure.

```
In [36]: def load_file(file):
             """Load and process a Worldbank File"""
             data = pd.read_csv(file, skiprows=4)
             data.set_index("Country Code", inplace=True)
             data.drop(non_countries, inplace=True)
             data = data[[str(year) for year in range(2010,2017)]]
             return data.mean(axis=1), data
```

```
In [37]: gdp_means, gdp = load_file("data/gdp-per-capita.csv")
```

```
In [38]: gdp_means.head()
```

```
Out[38]: Country Code
         ABW    24798.330391
         AFG      603.594404
         AGO     4134.962750
         ALB     4261.539631
         AND    39309.685362
         dtype: float64
```

```
In [39]: gdp.head()
```

```
Out[39]:                       2010          2011          2012          2013  \
         Country Code
         ABW           24271.940421  25324.720362           NaN           NaN
         AFG             553.300289    603.537023    669.009051    638.612543
         AGO            3529.053482   4299.012889   4598.249988   4804.616884
         ALB            4094.358832   4437.178068   4247.614308   4413.081697
         AND           39736.354063  41098.766942  38391.080867  40619.711298

                               2014          2015          2016
         Country Code
         ABW                    NaN           NaN           NaN
         AFG             629.345250    569.577923    561.778746
         AGO            4709.312024   3695.793748   3308.700233
         ALB            4578.666728   3934.895394   4124.982390
         AND           42294.994727  36038.267604  36988.622030
```

```
In [40]: population_means, population = load_file("data/population.csv")
```

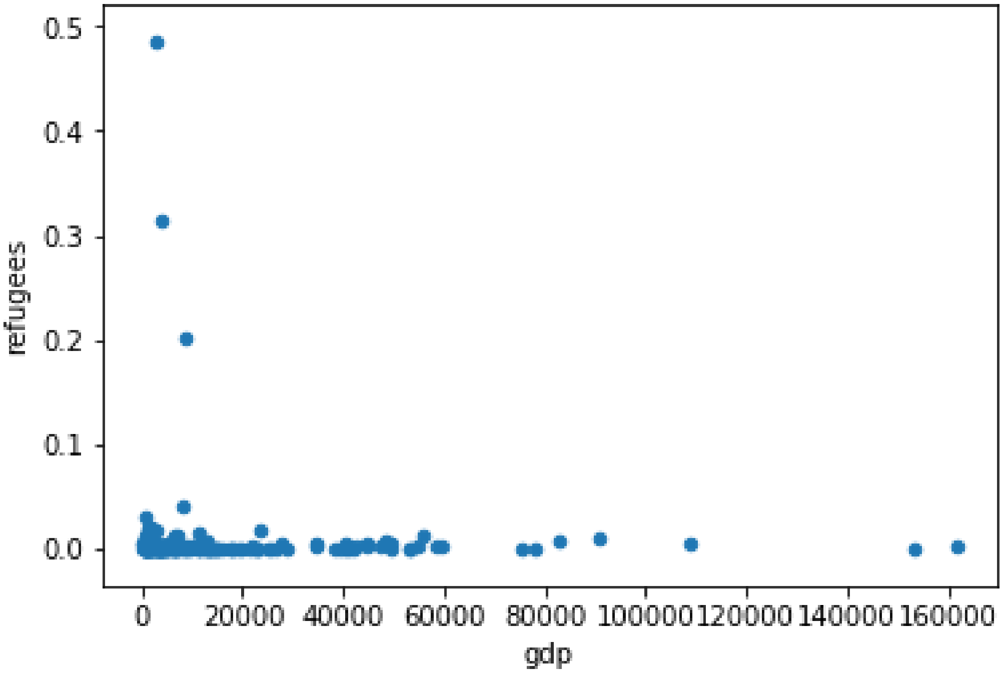
2.5 Creating the Plot

We now combine our three Series with means into one DataFrame and create our plot.

```
In [41]: data = pd.DataFrame({"gdp": gdp_means, "refugees": refugee_means/population_means}).d
```

(Here we loose some countries with missing data.)

```
In [42]: data.plot.scatter("gdp", "refugees")
         plt.show()
```



We can quickly find out who the three top countries are:

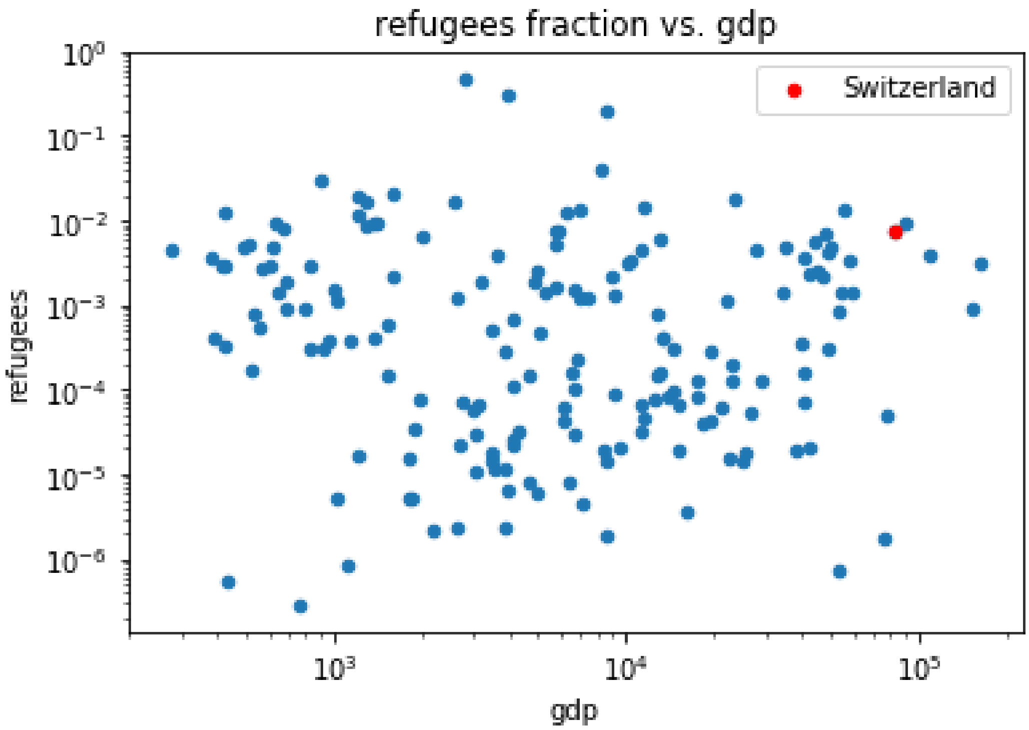
```
In [43]: data.where(data["refugees"]>0.1).dropna()
```

```
Out[43]:                           gdp  refugees
         Country Code
         JOR            3943.016044  0.313576
         LBN            8640.434951  0.202313
         PSE            2793.289949  0.484664
```

To improve readability:

- we switch to a log-log axis (we need to exclude countries with too small refugee numbers)
- we highlight one selected country
- We add a title

```
In [44]: ax = data[data.refugees > 1e-10].plot.scatter(y="refugees", x="gdp", loglog=True)
         ax = data.loc[["CHE"]].plot.scatter(y="refugees", x="gdp", ax=ax, color="r", label="S
         plt.title("refugees fraction vs. gdp")
         plt.show()
```



again we can print the info for one country

```
In [45]: data.loc["CHE"]
```

```
Out[45]: gdp           82883.020927
         refugees          0.007429
         Name: CHE, dtype: float64
```

2.5.1 Highlighting a Full Region

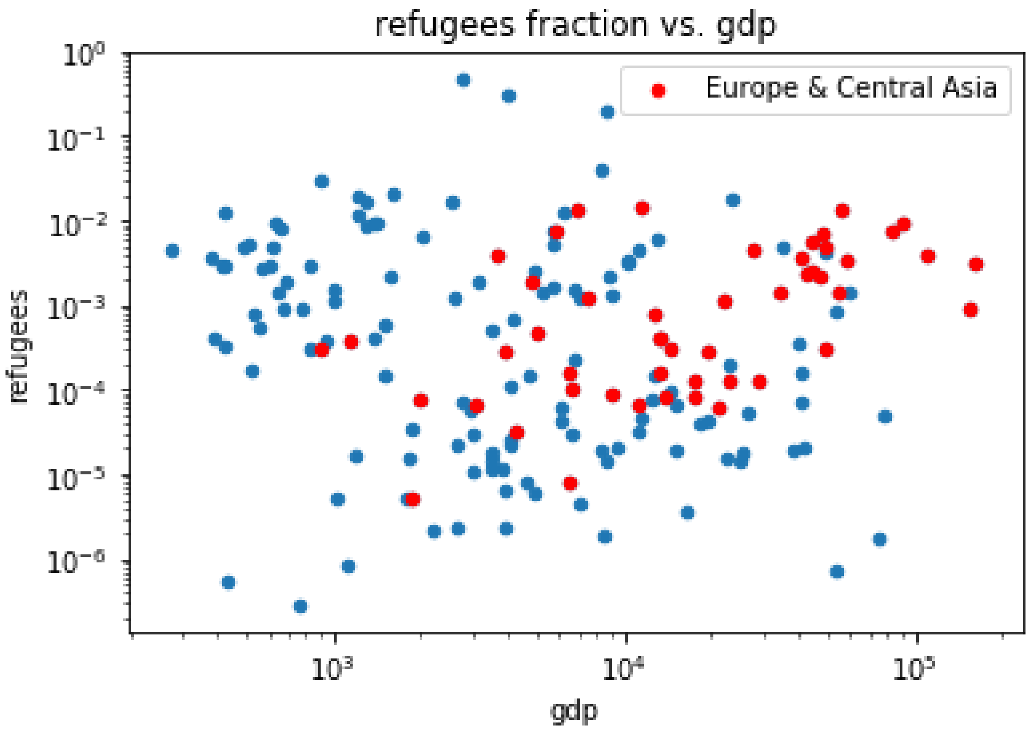
Based on th meta data provided by the World Bank, we can highlight a region

```
In [46]: europe = meta.loc[meta.Region == "Europe & Central Asia"].index
```

```
In [47]: europe[:10]
```

```
Out[47]: Index(['ALB', 'AND', 'ARM', 'AUT', 'AZE', 'BEL', 'BGR', 'BIH', 'BLR', 'CHE'], dtype='
```

```
In [48]: ax = data[data.refugees > 1e-10].plot.scatter(y="refugees", x="gdp", loglog=True)
         ax = data.loc[data.index.intersection(europe)].plot.scatter(y="refugees", x="gdp", ax
         plt.title("refugees fraction vs. gdp")
         plt.show()
```



(As we lost some countries with missing data when we called `dropna` above, we need the `data.index.intersection`-call to select only country codes really contained in our data.)

2.6 Fitting

We now look at a tiny subset of this data and look at ways to fit a function to it.

Scipy preparse a huge number of options, we will look at three options of increasing complexity and flexibility.

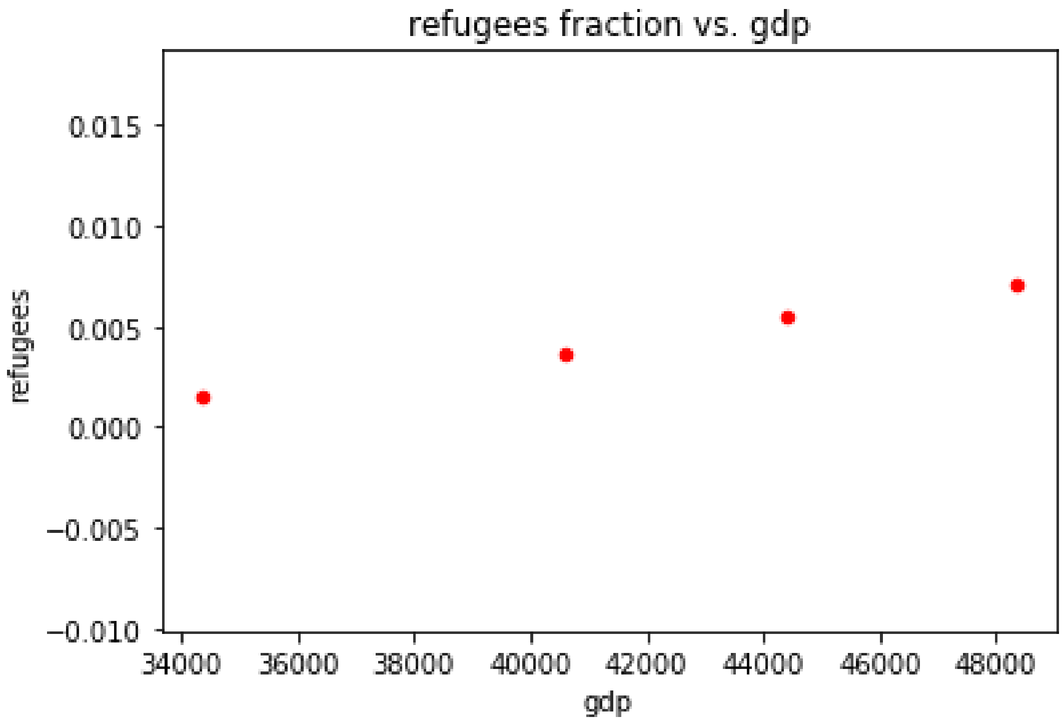
2.6.1 Preparations

first we select our subset

```
In [49]: europe_small = ['AUT',
           'DEU',
           'FRA',
           'ITA',
           #'CHE',
           #'ESP',
           #'PRT',
           #'BEL',
           #'LUX',
           #'NLD',
           #'NOR',
            #'DEN',
            #'SWE',
            #'FIN',
           ]
```

```
In [50]: data_eu = data.loc[europe_small].dropna()
```

```
In [51]: ax = data_eu.plot.scatter(y="refugees", x="gdp", color="r")
         plt.title("refugees fraction vs. gdp")
         plt.show()
```



and we create a vector with all the x values we will need to plot our fit result

```
In [52]: x = np.linspace(data_eu["gdp"].min(), data_eu["gdp"].max(), 100)
```

2.6.2 polyfit

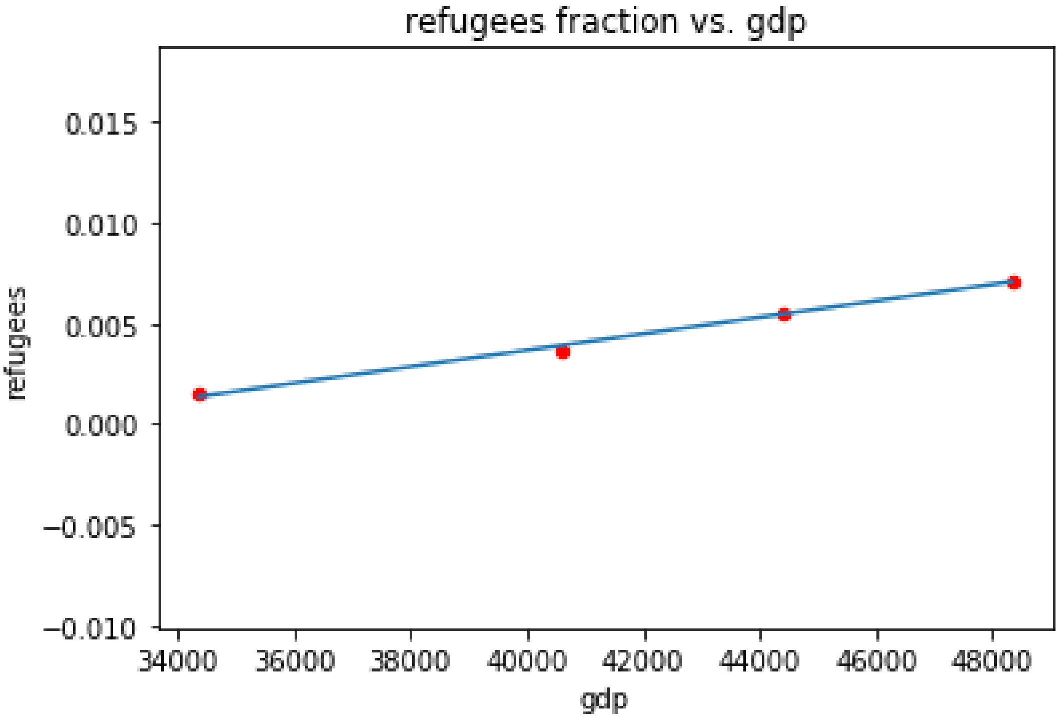
Polyfit is probably the easiest way to fit a polynome to given data.

```
In [53]: from scipy import polyfit, polyval
```

```
In [54]: res = polyfit(data_eu["gdp"], data_eu["refugees"],1)
         print(res)
```

```
[ 4.09014159e-07 -1.27066203e-02]
```

```
In [55]: ax = data_eu.plot.scatter(y="refugees", x="gdp", color="r")
         ax.plot(x, polyval(res, x))
         plt.title("refugees fraction vs. gdp")
         plt.show()
```



2.6.3 curve_fit

With `curve_fit` you can define a complex fit function.

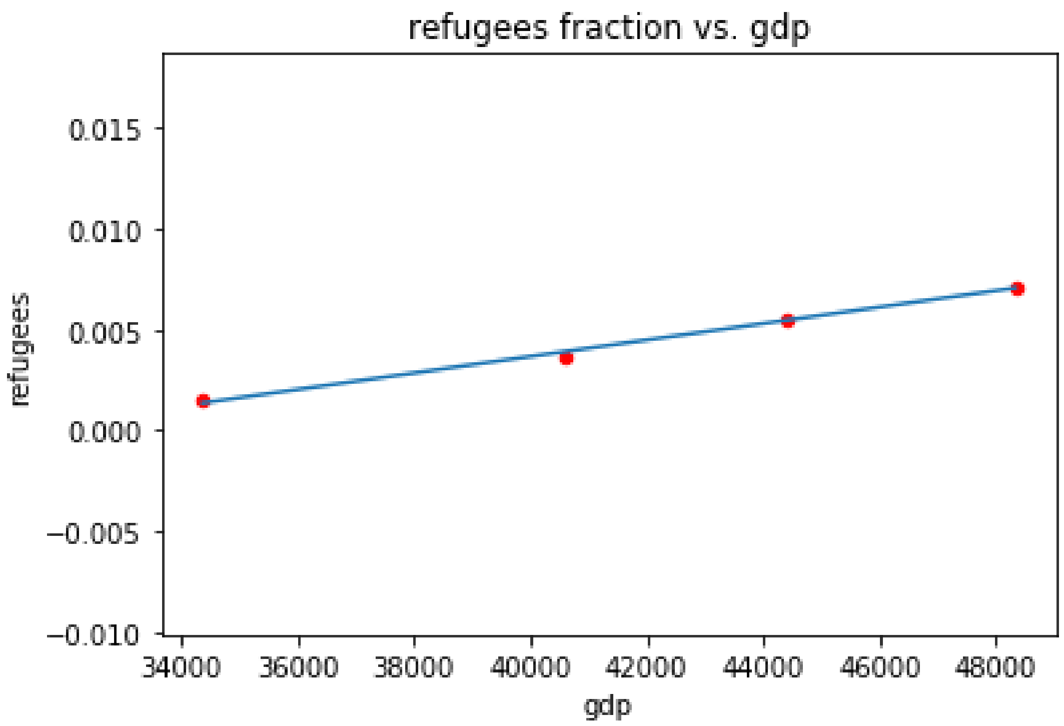
```
In [56]: from scipy.optimize import curve_fit
```

```
In [57]: def fit_function(x,b,c):
             return b*x+c
```

```
In [58]: res = curve_fit(fit_function, data_eu["gdp"], data_eu["refugees"])
         print(res)
```

```
(array([ 4.09014159e-07, -1.27066203e-02]), array([[ 3.74367147e-16, -1.56946192e-11],
       [-1.56946192e-11,  6.67866539e-07]]))
```

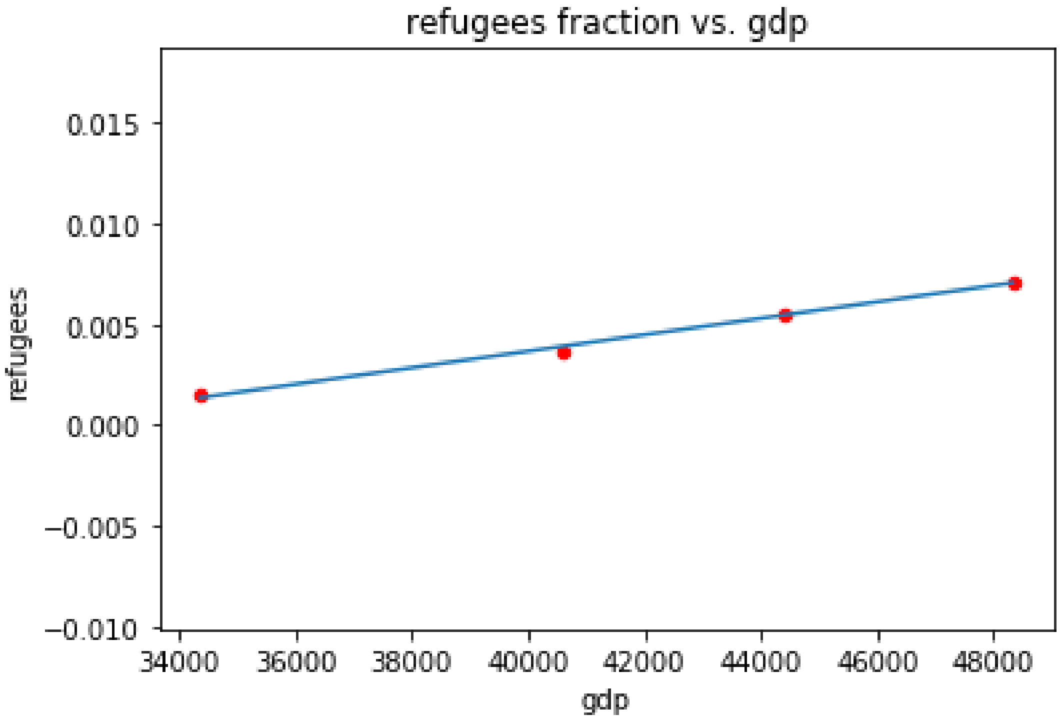
```
In [59]: ax = data_eu.plot.scatter(y="refugees", x="gdp", color="r")
         ax.plot(x, fit_function(x, *(res[0])))
         plt.title("refugees fraction vs. gdp")
         plt.show()
```



2.6.4 leastsq

Finally, least-squares allows you to even specify the cost function. With this you can factor in uncertainties or weights for your data points.

```
In [60]: from scipy.optimize import leastsq

In [61]: def fit_function(x, p):
             return x*p[0]+p[1]

In [62]: def error_function(params):
             return data_eu["refugees"] - fit_function(data_eu["gdp"], params)

In [63]: res = leastsq(error_function, [0,0])
         print(res)

(array([ 4.09014159e-07, -1.27066203e-02]), 2)


In [64]: ax = data_eu.plot.scatter(y="refugees", x="gdp", color="r")
         ax.plot(x, fit_function(x, res[0]))
         plt.title("refugees fraction vs. gdp")
         plt.show()
```


refugees fraction vs. gdp
refugees
gdp
0.015
0.010
0.005
0.000
−0.005
−0.010
34000
36000
38000
40000
42000
44000
46000
48000