Chapter 10: Plotting and diagram visualization

Matplotlib is the most popular 2D plotting library in Python. Using matplotlib, you can create pretty much any type of plot.

Pandas has **tight integration** with *matplotlib*.

How to install matplotlib?

If you have Anaconda installed, then matplotlib was already installed together with it.

If you have a standalone Python3 and Jupyter Notebook installation, open a command prompt / terminal and type in:

```
pip3 install matplotlib
```

How to use matplotlib?

We will use the *pyplot module* inside the matlplotlib package for plotting. You can simply import this module as usual. It is usually aliased with the `plt` abbreviation:

```python
import matplotlib.pyplot as plt
```

The dataset

Let's use the *World Countries datatset*. For each country the following information is given:

1. country name,
2. region name,
3. population,
4. area (in mi^2),
5. GDP ($ per capita),
6. Literacy (%)

The dataset is given in the `data/countries_world.csv` file. The used delimiter is the semicolon (`;`) character.

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Special Jupyter Notebook command, so the plots by matplotlib will be display i
nside the Jupyter Notebook
%matplotlib inline

countries = pd.read_csv('../data/countries_world.csv', delimiter = ';')
countries.columns = ['country', 'region', 'population', 'area', 'gdp', 'literac
y']
display(countries)
```

	country	region	population	area	gdp	literacy
0	Afghanistan	ASIA (EX. NEAR EAST)	31056997	647500	700.0	36.0
1	Albania	EASTERN EUROPE	3581655	28748	4500.0	86.5
2	Algeria	NORTHERN AFRICA	32930091	2381740	6000.0	70.0
3	American Samoa	OCEANIA	57794	199	8000.0	97.0
4	Andorra	WESTERN EUROPE	71201	468	19000.0	100.0
...	...	...	...	...	...	...
222	West Bank	NEAR EAST	2460492	5860	800.0	NaN
223	Western Sahara	NORTHERN AFRICA	273008	266000	NaN	NaN
224	Yemen	NEAR EAST	21456188	527970	800.0	50.2
225	Zambia	SUB-SAHARAN AFRICA	11502010	752614	800.0	80.6
226	Zimbabwe	SUB-SAHARAN AFRICA	12236805	390580	1900.0	90.7

227 rows × 6 columns

Data source: [US Government (https://gsociology.icaap.org/dataupload.html)](https://gsociology.icaap.org/dataupload.html)

Lets take just the top 50 countries by area, so visualization will be easier to overview in the following tasks:

In [2]:

```
countries50 = countries.sort_values(by = 'area', ascending = False).head(50)
display(countries50)
```

	country	region	population	area	gdp	literacy
169	Russia	C.W. OF IND. STATES	142893540	17075200	8900.0	99.6
36	Canada	NORTHERN AMERICA	33098932	9984670	29800.0	97.0
214	United States	NORTHERN AMERICA	298444215	9631420	37800.0	97.0
42	China	ASIA (EX. NEAR EAST)	1313973713	9596960	5000.0	90.9
27	Brazil	LATIN AMER. & CARIB	188078227	8511965	7600.0	86.4
...	...	...	...	...	...	...
124	Madagascar	SUB-SAHARAN AFRICA	18595469	587040	800.0	68.9
107	Kenya	SUB-SAHARAN AFRICA	34707817	582650	1000.0	85.1
69	France	WESTERN EUROPE	60876136	547030	27600.0	99.0
224	Yemen	NEAR EAST	21456188	527970	800.0	50.2
201	Thailand	ASIA (EX. NEAR EAST)	64631595	514000	7400.0	92.6

Plotting

Plots can be generated with the `plot()` function of a Pandas *DataFrame* (table) or *Series* (column). The most important parameter of the function is the `kind` parameter, which defines the type of plot to be generated. Supported kinds are (non-exhaustive list):

- `line`
- `bar` (vertical bar)
- `barh` (horizontal bar)
- `scatter`
- `hist` (histogram)
- `box` (boxplot
- `pie`

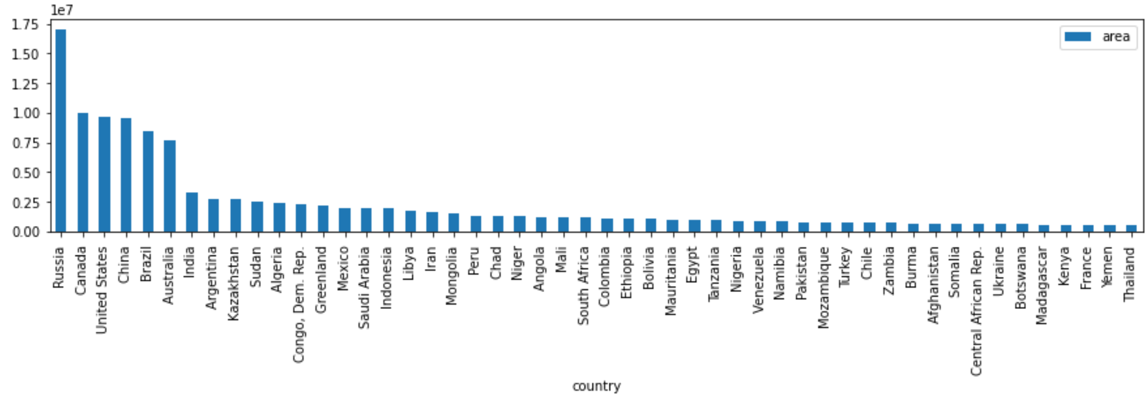
After a plot is generated, it can be displayed by the `show()` function of the `matplotlib.pyplot` module.

Vertical bar plot

Display a bar plot on the area of the selected 50 largest countries.

In [3]:

```
countries50.plot(kind='bar', x='country', y='area', figsize = [15, 3])
plt.show() # matplotlib.pyplot was imported as plt
```

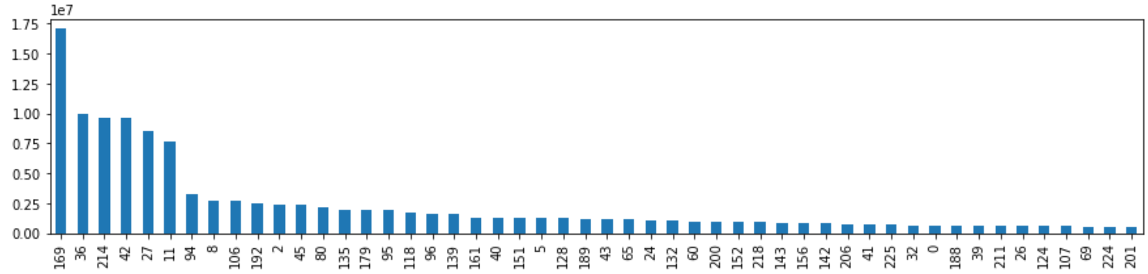


The size of the diagram can be configured with the `figsize` parameter. The size is given in inches (1 inch = 2.54 centimeters).
The default size is `[6.4, 4.8]` .

The bar diagram can be created directly on the selected *Series* (column of data). In this case the Series will be placed along axis Y, while the horizontal axis X will become the index of the *DataFrame*.

In [4]:

```
countries50['area'].plot(kind='bar', figsize = [15, 3])
plt.show()
```



The index column can be modified through the `set_index` function (see Chapter 7 for more details) of the *DataFrame* and a **new** *DataFrame* is created so:

In [5]:

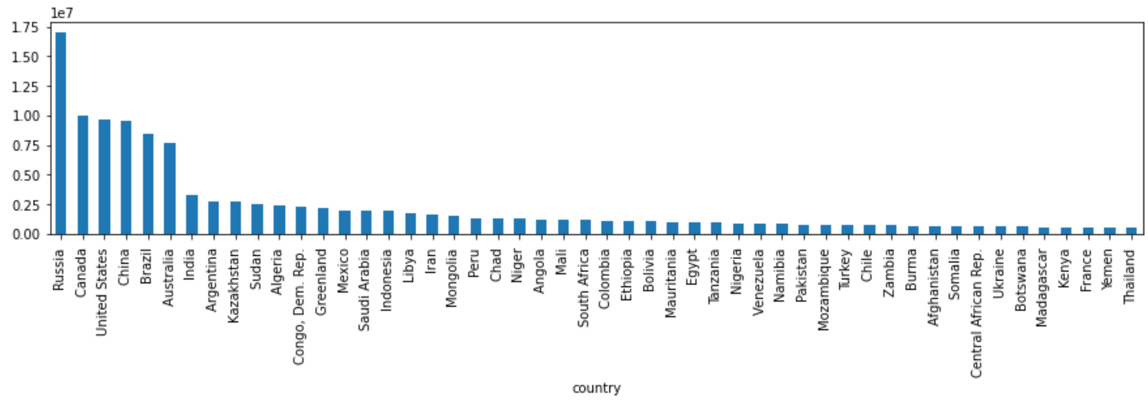
```
countries50_indexed = countries50.set_index('country')
display(countries50_indexed)
```

country	region	population	area	gdp	literacy
Russia	C.W. OF IND. STATES	142893540	17075200	8900.0	99.6
Canada	NORTHERN AMERICA	33098932	9984670	29800.0	97.0
United States	NORTHERN AMERICA	298444215	9631420	37800.0	97.0
China	ASIA (EX. NEAR EAST)	1313973713	9596960	5000.0	90.9
Brazil	LATIN AMER. & CARIB	188078227	8511965	7600.0	86.4
...	...	...	...	...	...
Madagascar	SUB-SAHARAN AFRICA	18595469	587040	800.0	68.9
Kenya	SUB-SAHARAN AFRICA	34707817	582650	1000.0	85.1
France	WESTERN EUROPE	60876136	547030	27600.0	99.0
Yemen	NEAR EAST	21456188	527970	800.0	50.2
Thailand	ASIA (EX. NEAR EAST)	64631595	514000	7400.0	92.6

Creating the bar plot from the `countries50_indexed` *DataFrame* will display the country names as labels correctly.

In [6]:

```
countries50_indexed['area'].plot(kind='bar', figsize = [15, 3])
plt.show()
```

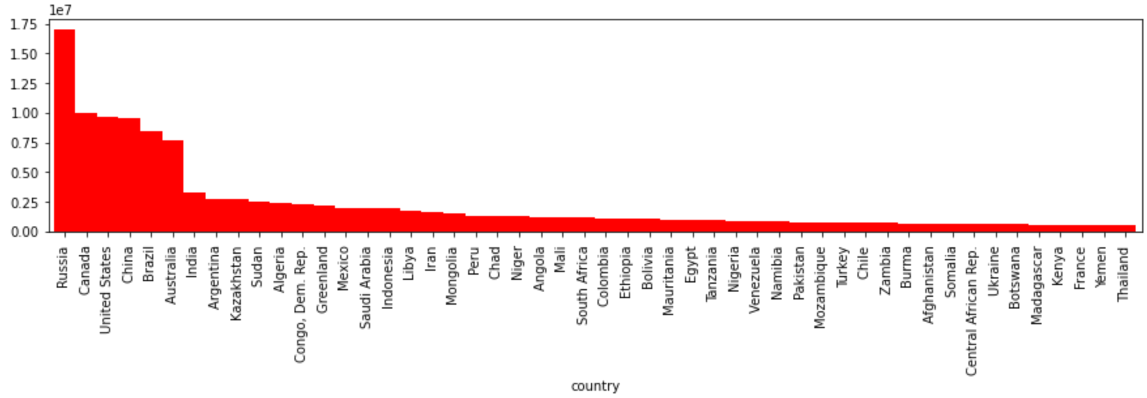


Visual tuning

The color of the bars can be defined with the `color` parameter. The width of the bars is set by the `width` parameter, 1.0 meaning *100%*.

In [7]:

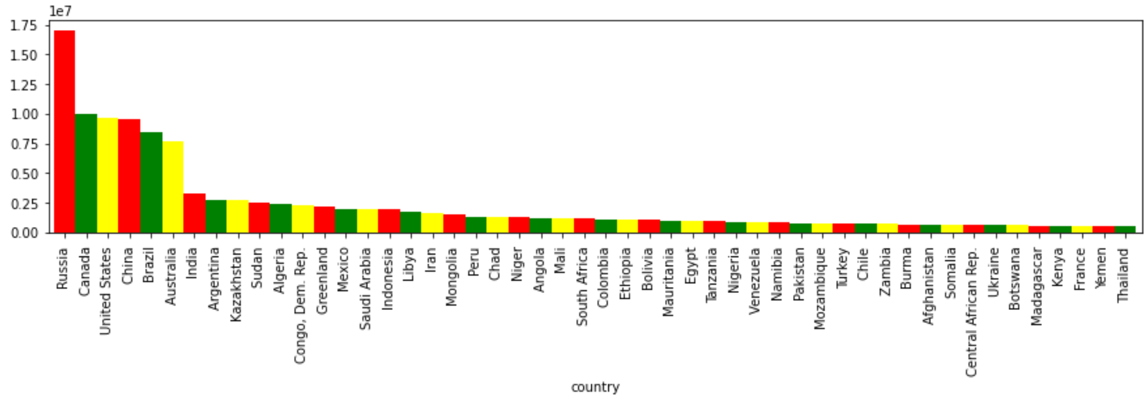
```
countries50_indexed['area'].plot(kind='bar', figsize = [15, 3], color = 'red', w
idth = 1.0)
plt.show()
```



Multiple colors can be passed in a list.

In [8]:

```
countries50_indexed['area'].plot(kind='bar', figsize = [15, 3], color = ['red',
'green', 'yellow'], width = 1.0)
plt.show()
```

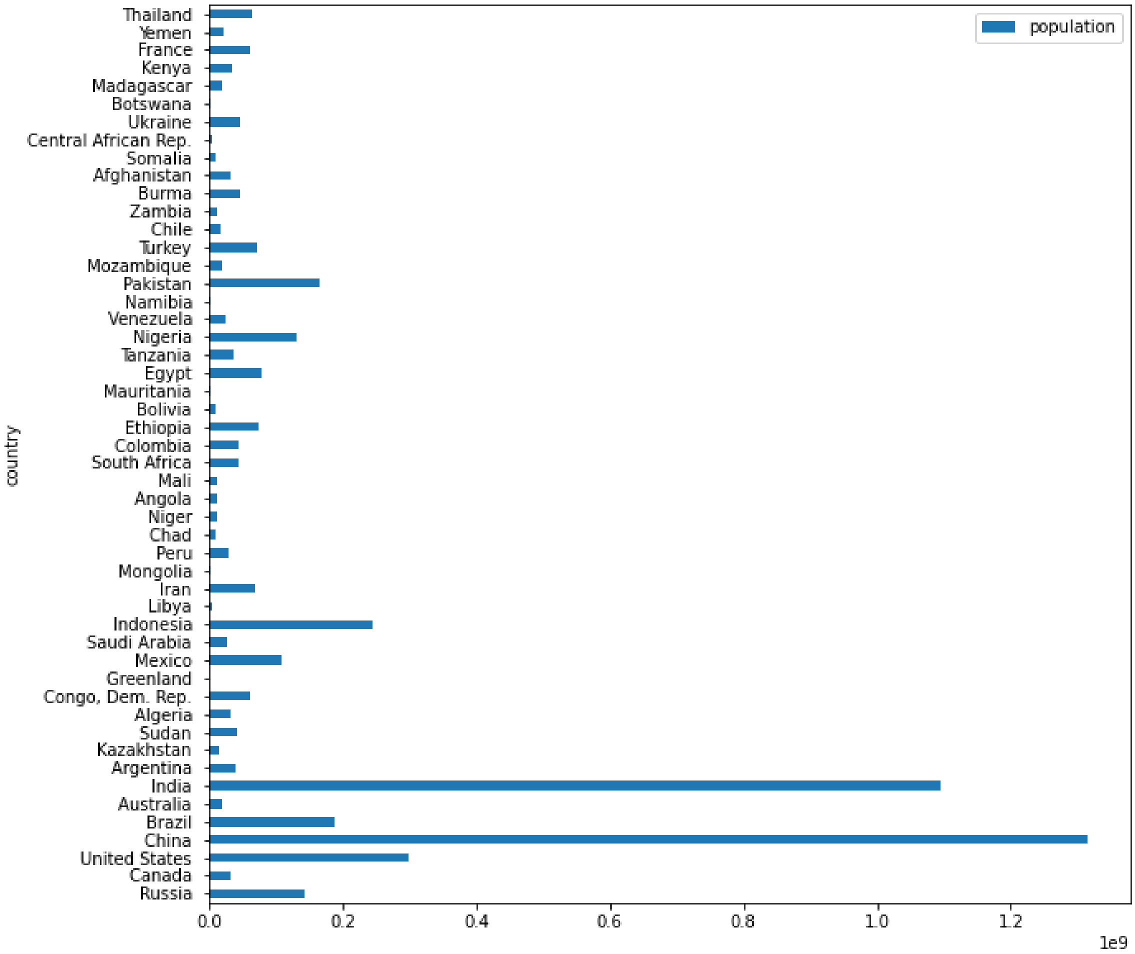


Horizontal bar plot

Display a horizontal bar plot on the population of the selected 50 largest countries.

In [9]:

```
countries50.plot(kind='barh', x='country', y='population', figsize = [10, 10])
plt.show()
```

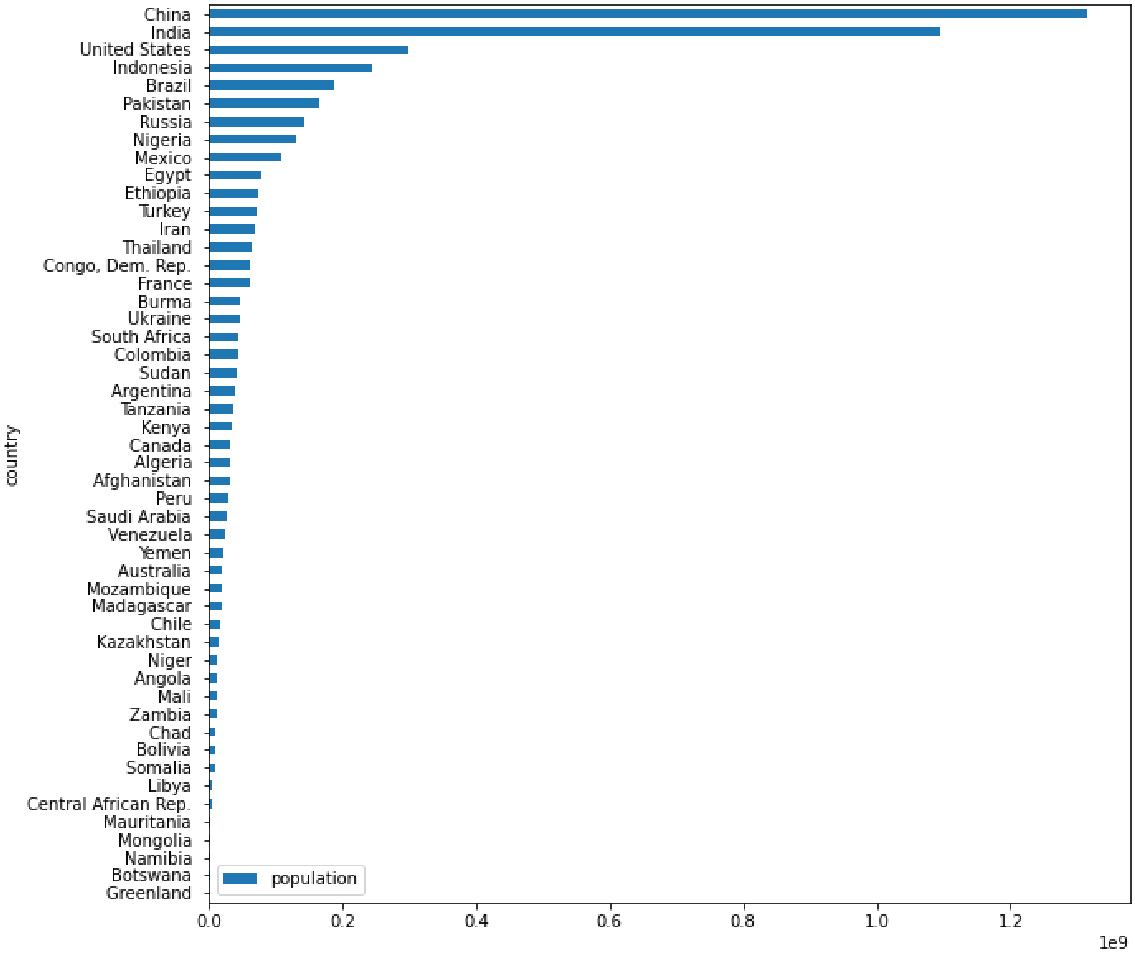


Note that for the horizontal bar plot, the *axis X* is the vertical axis, and *axis Y* is the horizontal axis. It is defined by this was, so only the `kind` parameter of the `plot()` function has to be changed when switching to a different type of diagram.

Before visualizing the data, sort it by the column population, instead of the default area.

In [10]:

```
countries50.sort_values(by = 'population').plot(kind='barh', x='country', y='population', figsize = [10, 10])
plt.show()
```



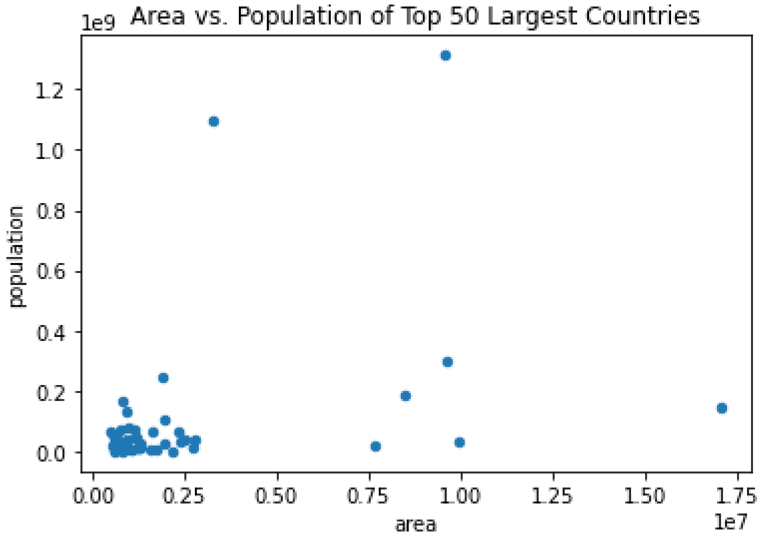
Scatter plot

Display a scatter plot on the correlation of the area and the population columns of the selected 50 largest countries.

Question: What correlation can be expected between these 2 attributes of countries?

In [11]:

```
countries50.plot(kind='scatter', x='area', y='population', title='Area vs. Popul
ation of Top 50 Largest Countries')
plt.show()
```

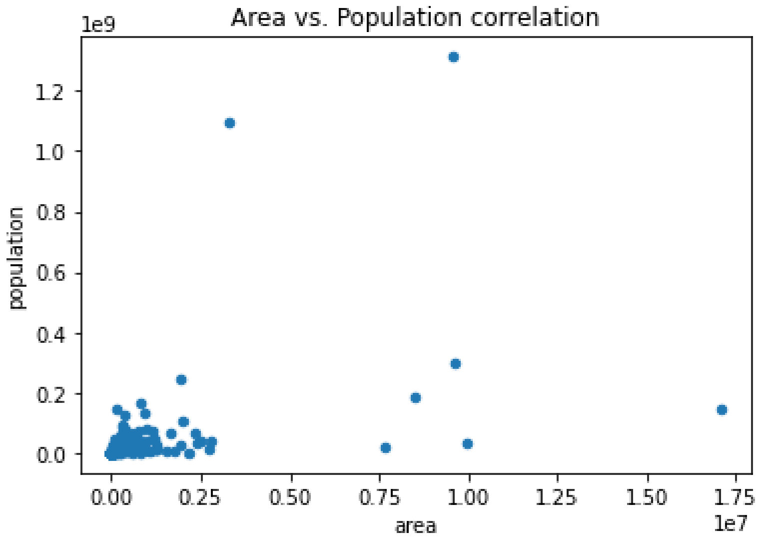


A title can be given to be displayed above the generated diagram with the `title` parameter.

Extend the scatter plot for all countries in the dataset.

In [12]:

```
countries.plot(kind='scatter', x='area', y='population', title='Area vs. Populat
ion correlation')
plt.show()
```

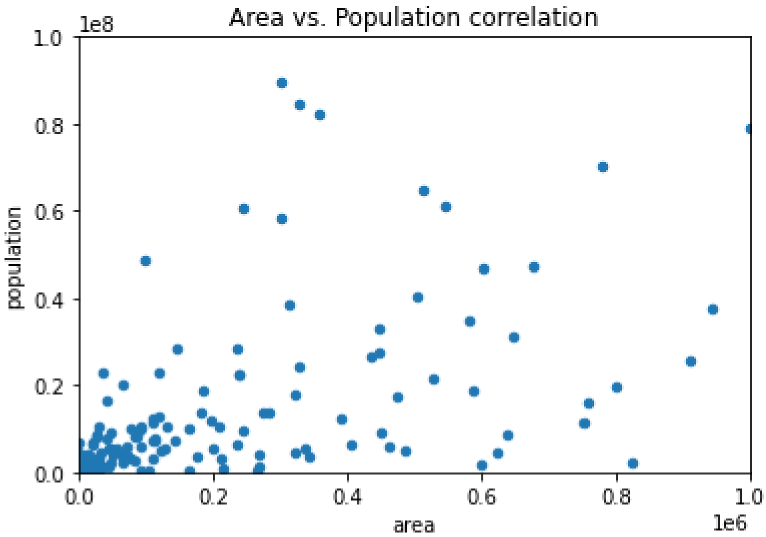


As we can observe there is a moderately strong correlation between area and population, which matches our expectation.

The limits of the X and Y axes can be configured with the `xlim` and `ylim` parameters, so the *outliers* can be excluded from the visualization. Both a minimum and a maximum boundary can be given, as a tuple.

In [13]:

```
countries.plot(kind='scatter', x='area', y='population', title='Area vs. Populat
ion correlation', xlim=(0, 1e6), ylim=(0, 1e8))
plt.show()
```



Short outlook on correlation *(optional)*

The correlation matrix between *Series* of a *Pandas DataFrame* can be generated with the `corr()` function:

In [14]:

```
display(countries.corr())
```

	population	area	gdp	literacy
population	1.000000	0.469985	-0.039324	-0.043481
area	0.469985	1.000000	0.072185	0.035994
gdp	-0.039324	0.072185	1.000000	0.513144
literacy	-0.043481	0.035994	0.513144	1.000000

Or just for 2 selected Series:

In [15]:

```
print(countries['area'].corr(countries['population']))
```

```
0.46998508371848174
```

Every correlation has two qualities: *strength* and *direction*. The direction of a correlation is either positive or negative. When two variables have a positive correlation, it means the variables move in the same direction. This means that as one variable increases, so does the other one. In a negative correlation, the variables move in inverse, or opposite, directions. In other words, as one variable increases, the other variable decreases.

We determine the strength of a relationship between two correlated variables by looking at the numbers. A correlation of 0 means that no relationship exists between the two variables, whereas a correlation of 1 indicates a perfect positive relationship. It is uncommon to find a perfect positive relationship in the real world.

The further away from 1 that a positive correlation lies, the weaker the correlation. Similarly, the further a negative correlation lies from -1, the weaker the correlation. The following guidelines are useful when determining the strength of a positive correlation:

- 1: perfect positive correlation
- .70 to .99: very strong positive relationship
- .40 to .69: strong positive relationship
- .30 to .39: moderate positive relationship
- .20 to .29: weak positive relationship
- .01 to .19: no or negligible relationship
- 0: no relationship exists

Question: which series of the dataframe show strong correlation?

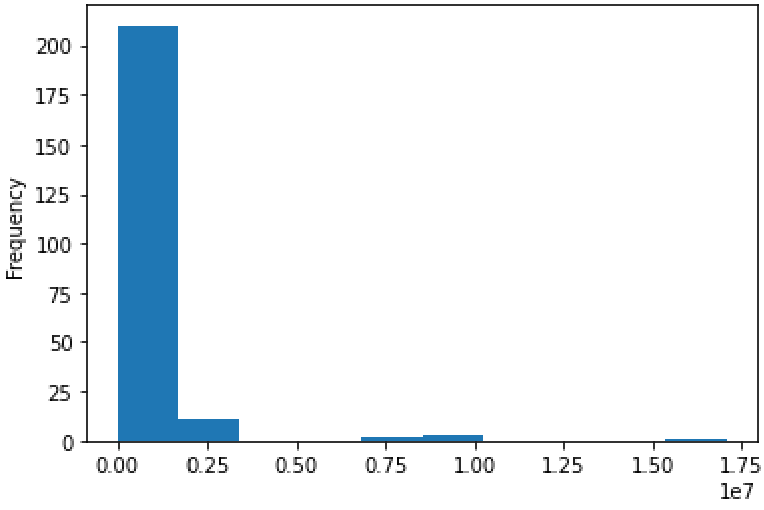
Histogram

A histogram is an accurate representation of the distribution of numerical data. It differs from a bar graph, in the sense that a bar graph relates two variables, but a histogram relates only one.

Display a histogram on the area of the selected 50 countries.

In [16]:

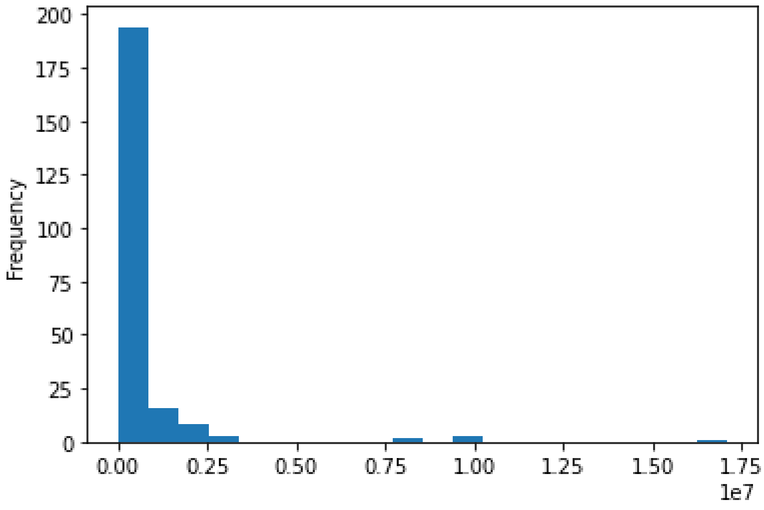
```
countries['area'].plot(kind='hist')
plt.show()
```



The number of columns (called *bins* or *buckets*) in the histrogram can be configured with the `bins` parameter.

In [17]:

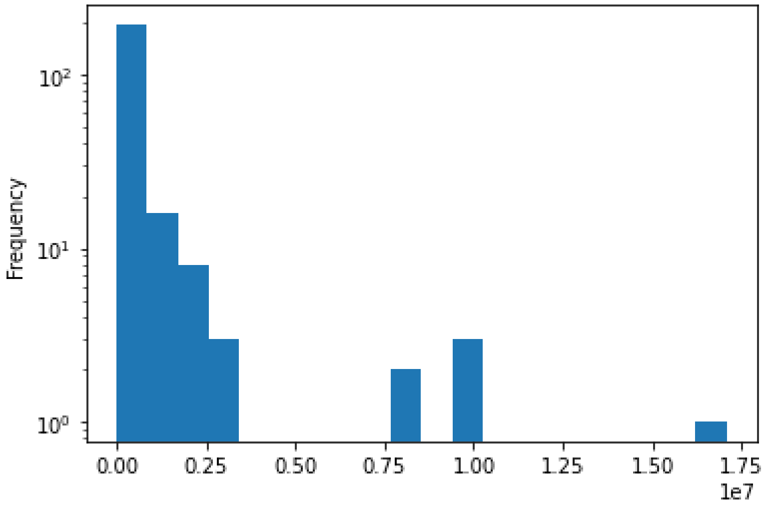
```
countries['area'].plot(kind='hist', bins=20)
plt.show()
```



Extend the histogram to cover all countries in the dataset. Apply a logarithmic scale with the `logx / logy` parameter.

In [18]:

```
countries['area'].plot(kind='hist', bins=20, logy=True)
plt.show()
```



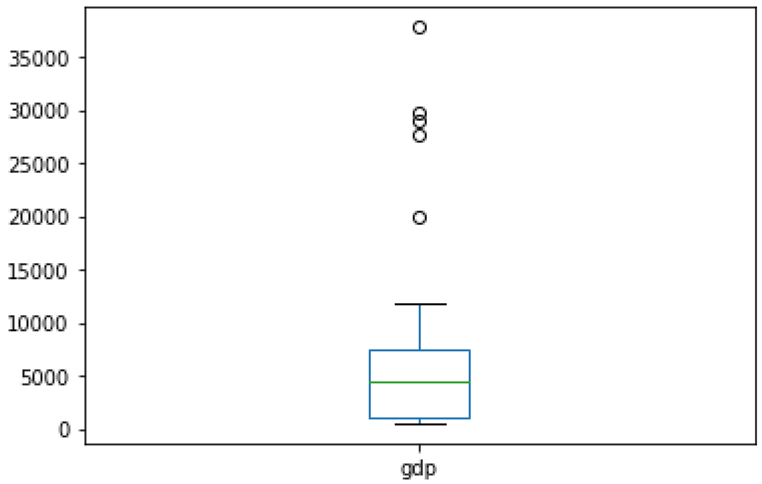
Boxplot

In descriptive statistics, a *boxplot* is a method for graphically depicting groups of numerical data through their quartiles.

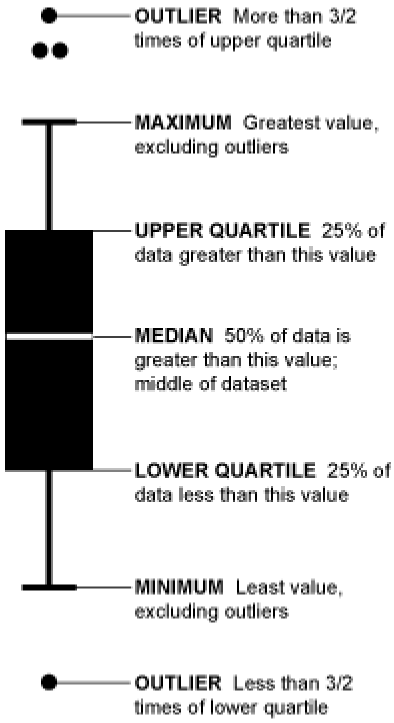
Display a boxplot on the GDP of the selected 50 largest countries.

In [19]:

```
countries50['gdp'].plot(kind='box')
plt.show()
```



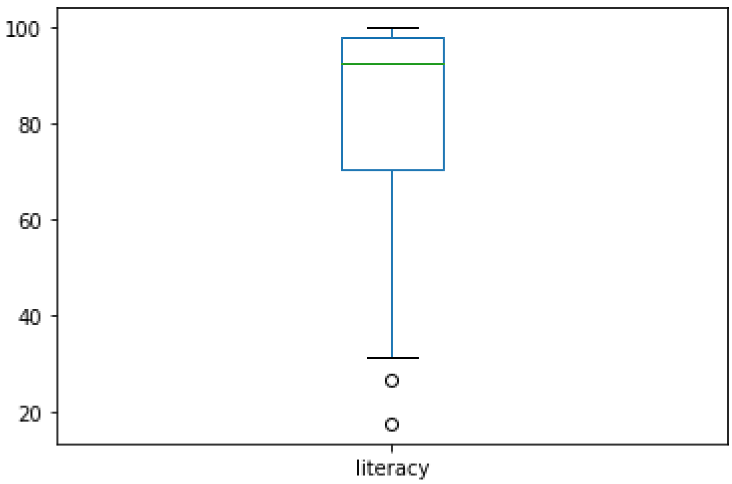
Explaining the graphical visualization of a boxplot:



Task: Display a boxplot on the literacy of all countries! What can we state based on the diagram?

In [20]:

```
countries['literacy'].plot(kind='box')
plt.show()
```

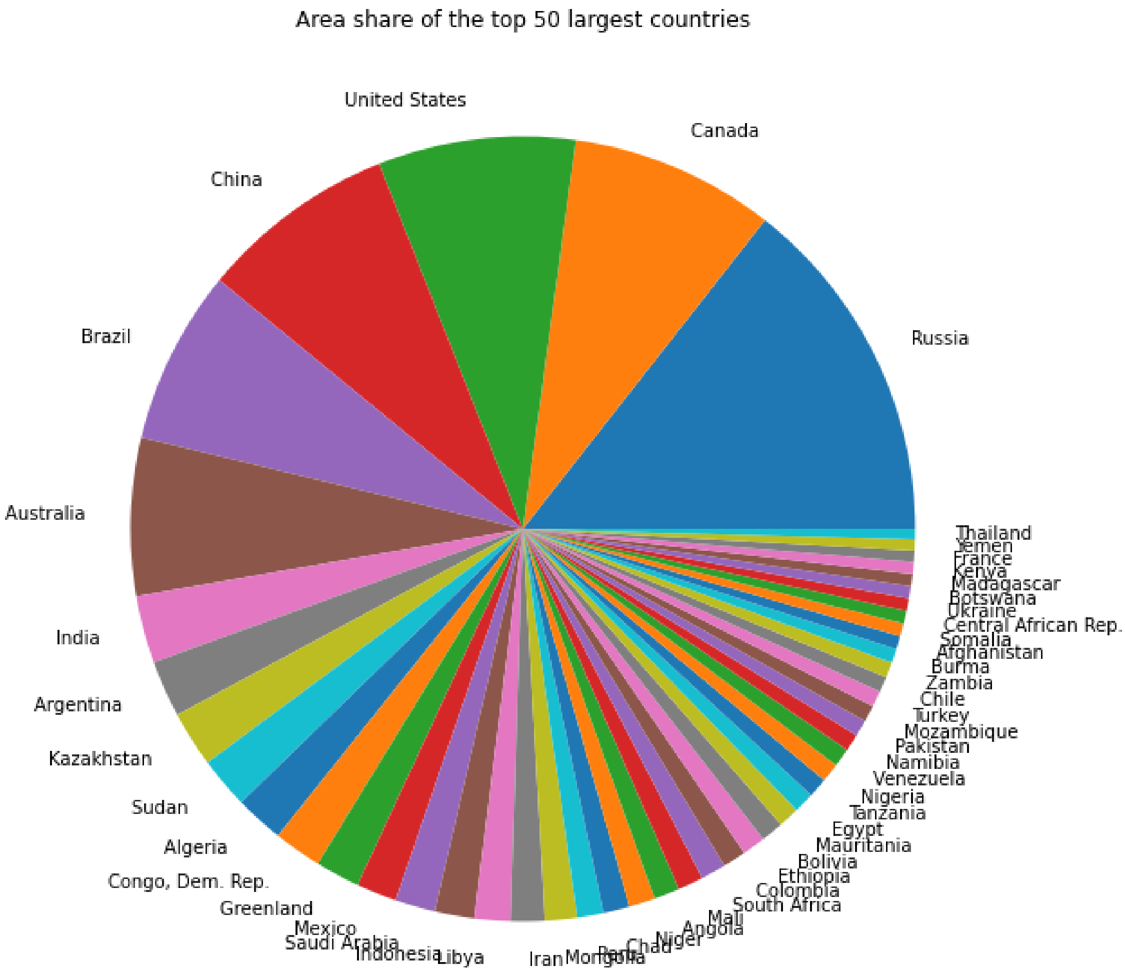


Pie chart

Display a pie chart on the area share of the selected 50 largest countries. Since we are creating this plot on the `area` *Series*, we use the `countries50_indexed` DataFrame, which was indexed with the country names, so the labels will contain them instead of numerical indices.

In [21]:

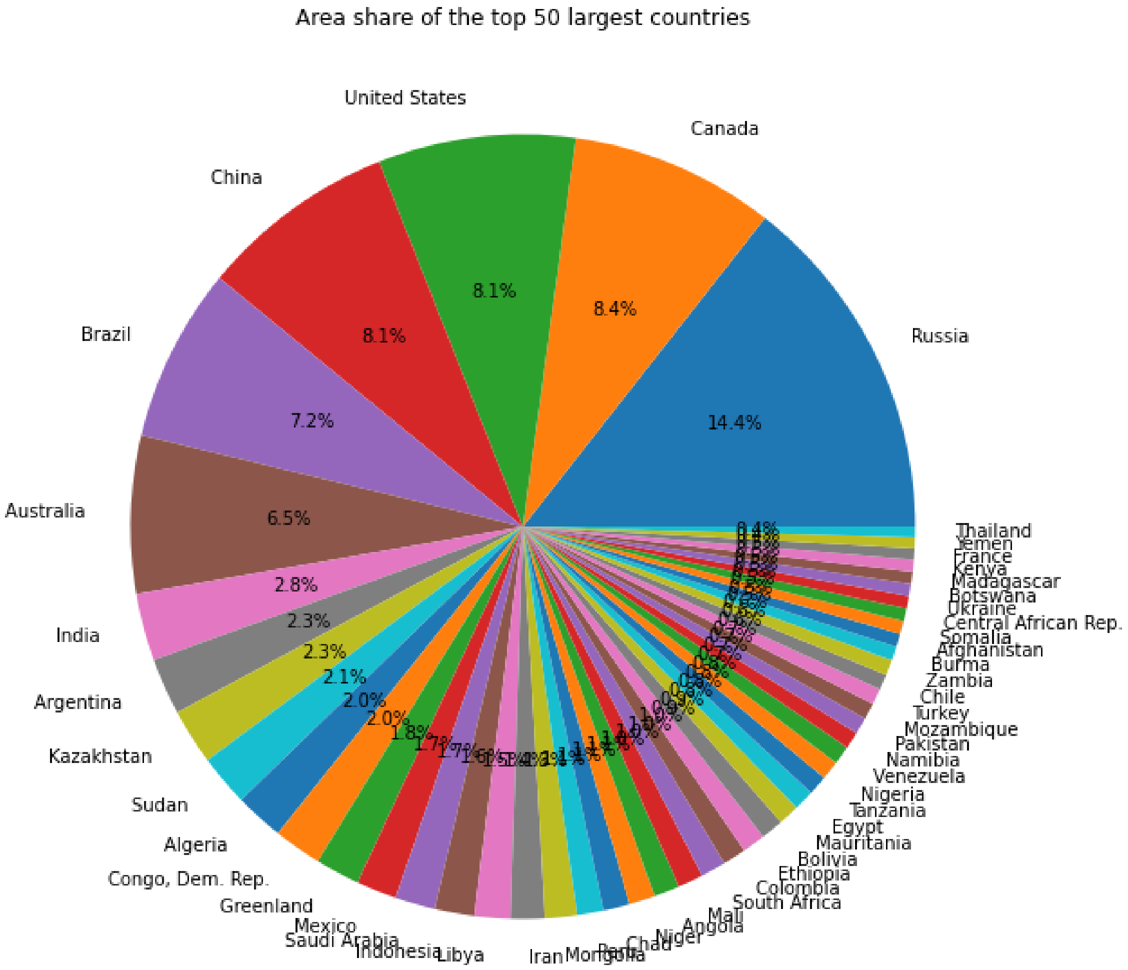
```
countries50_indexed['area'].plot(kind='pie', figsize=[10,10], label="", title="A
rea share of the top 50 largest countries")
plt.show()
```



Percentages for each slice can be displayed with the `autopct` parameter:

In [22]:

```
countries50_indexed['area'].plot(kind='pie', figsize=[10,10], autopct='%.1f%%',
label="", title="Area share of the top 50 largest countries")
plt.show()
```

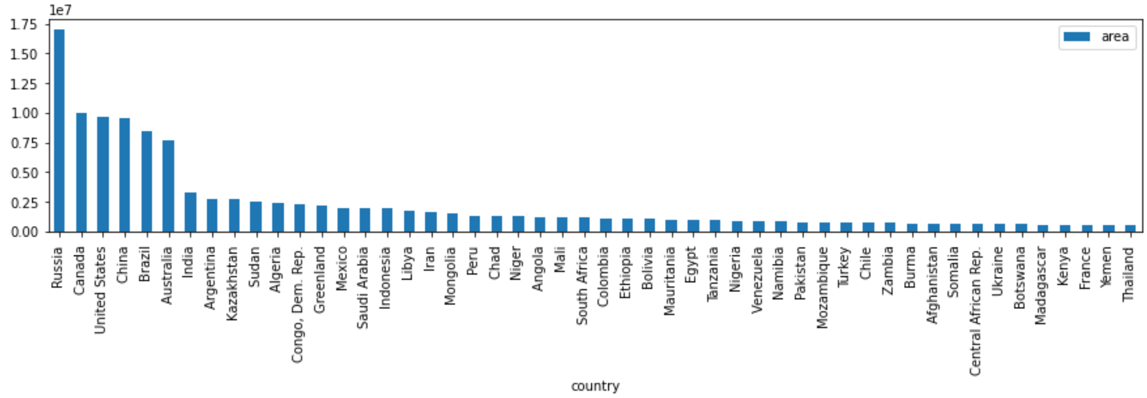


Saving a plot to file

Intead of using the `show()` function of the `matplotlib.pyplot` module, the `savefig()` function can also be used to export and save a created plot to an external file.

In [23]:

```
countries50.plot(kind='bar', x='country', y='area', figsize = [15, 3])
plt.savefig('10_country_area.png')
```



Hint: look for the created file right next this Jupyter Notebook file on your computer.

Time Series Analysis

Read the *Population History dataset* from the `data/population_world.csv` file, which contains the population data for all countries between the years 1950 and 2019. All together the dataset contains 239 countries (or territories), 70 years of data, so all together 16,730 rows of data.

Each row stores the following data:

1. location (country or region),
2. year,
3. male population,
4. female population,
5. total population,
6. population density.

The used delimiter is the semicolon (`;`) character.

In [24]:

```
population_history = pd.read_csv('../data/population_history.csv', delimiter =
';')
display(population_history)
```

	Country	Year	PopMale	PopFemale	PopTotal	PopDensity
0	Afghanistan	1950	4099.243	3652.874	7752.117	11.874
1	Afghanistan	1951	4134.756	3705.395	7840.151	12.009
2	Afghanistan	1952	4174.450	3761.546	7935.996	12.156
3	Afghanistan	1953	4218.336	3821.348	8039.684	12.315
4	Afghanistan	1954	4266.484	3884.832	8151.316	12.486
...	...	...	...	...	...	...
16725	Zimbabwe	2015	6568.778	7245.864	13814.642	35.711
16726	Zimbabwe	2016	6674.206	7356.132	14030.338	36.268
16727	Zimbabwe	2017	6777.054	7459.545	14236.599	36.801
16728	Zimbabwe	2018	6879.119	7559.693	14438.812	37.324
16729	Zimbabwe	2019	6983.353	7662.120	14645.473	37.858

16730 rows × 6 columns

Data source: [United Nations (https://www.un.org/development/desa/pd/)](https://www.un.org/development/desa/pd/)

Display the countries in the dataset:

In [25]:

```
print(population_history['Country'].unique())
```

```
['Afghanistan' 'Albania' 'Algeria' 'American Samoa' 'Andean Communit
y'
 'Andorra' 'Angola' 'Anguilla' 'Antigua and Barbuda' 'Argentina' 'Ar
menia'
 'Aruba' 'Australia' 'Australia/New Zealand' 'Austria' 'Azerbaijan'
 'Bahamas' 'Bahrain' 'Bangladesh' 'Barbados' 'Belarus' 'Belgium' 'Be
lize'
 'Benin' 'Bermuda' 'Bhutan' 'Bolivia (Plurinational State of)'
 'Bonaire, Sint Eustatius and Saba' 'Bosnia and Herzegovina' 'Botswa
na'
 'Brazil' 'British Virgin Islands' 'Brunei Darussalam' 'Bulgaria'
 'Burkina Faso' 'Burundi' "Côte d'Ivoire" 'Cabo Verde' 'Cambodia'
 'Cameroon' 'Canada' 'Cayman Islands' 'Central African Republic' 'Ch
ad'
 'Channel Islands' 'Chile' 'China' 'China, Hong Kong SAR'
 'China, Macao SAR' 'China, Taiwan Province of China' 'Colombia' 'Co
moros'
 'Congo' 'Cook Islands' 'Costa Rica' 'Croatia' 'Cuba' 'Curaçao' 'Cyp
rus'
 'Czechia' "Dem. People's Republic of Korea"
 'Democratic Republic of the Congo' 'Denmark' 'Djibouti' 'Dominica'
 'Dominican Republic' 'Ecuador' 'Egypt' 'El Salvador' 'Equatorial Gu
inea'
 'Eritrea' 'Estonia' 'Eswatini' 'Ethiopia' 'Falkland Islands (Malvin
as)'
 'Faroe Islands' 'Fiji' 'Finland' 'France' 'French Guiana'
 'French Polynesia' 'Gabon' 'Gambia' 'Georgia' 'Germany' 'Ghana'
 'Gibraltar' 'Greece' 'Greenland' 'Grenada' 'Guadeloupe' 'Guam'
 'Guatemala' 'Guinea' 'Guinea-Bissau' 'Guyana' 'Haiti' 'Holy See'
 'Honduras' 'Hungary' 'Iceland' 'India' 'Indonesia'
 'Iran (Islamic Republic of)' 'Iraq' 'Ireland' 'Isle of Man' 'Israe
l'
 'Italy' 'Jamaica' 'Japan' 'Jordan' 'Kazakhstan' 'Kenya' 'Kiribati'
 'Kuwait' 'Kyrgyzstan' "Lao People's Democratic Republic" 'Latvia'
 'Lebanon' 'Lesotho' 'Liberia' 'Libya' 'Liechtenstein' 'Lithuania'
 'Luxembourg' 'Madagascar' 'Malawi' 'Malaysia' 'Maldives' 'Mali' 'Ma
lta'
 'Marshall Islands' 'Martinique' 'Mauritania' 'Mauritius' 'Mayotte'
 'Melanesia' 'Mexico' 'Micronesia' 'Micronesia (Fed. States of)' 'Mo
naco'
 'Mongolia' 'Montenegro' 'Montserrat' 'Morocco' 'Mozambique' 'Myanma
r'
 'Namibia' 'Nauru' 'Nepal' 'Netherlands' 'New Caledonia' 'New Zealan
d'
 'Nicaragua' 'Niger' 'Nigeria' 'Niue' 'North Macedonia'
 'Northern Mariana Islands' 'Norway' 'Oman' 'Pakistan' 'Palau' 'Pana
ma'
 'Papua New Guinea' 'Paraguay' 'Peru' 'Philippines' 'Poland' 'Polyne
sia'
 'Portugal' 'Puerto Rico' 'Qatar' 'Réunion' 'Republic of Korea'
 'Republic of Moldova' 'Romania' 'Russian Federation' 'Rwanda'
 'Saint Barthélemy' 'Saint Helena' 'Saint Kitts and Nevis' 'Saint Lu
cia'
 'Saint Martin (French part)' 'Saint Pierre and Miquelon'
 'Saint Vincent and the Grenadines' 'Samoa' 'San Marino'
 'Sao Tome and Principe' 'Saudi Arabia' 'Senegal' 'Serbia' 'Seychell
es'
 'Sierra Leone' 'Singapore' 'Sint Maarten (Dutch part)' 'Slovakia'
 'Slovenia' 'Solomon Islands' 'Somalia' 'South Sudan' 'Spain' 'Sri L
anka'
 'State of Palestine' 'Sudan' 'Suriname' 'Sweden' 'Switzerland'
```

```
 'Syrian Arab Republic' 'Tajikistan' 'Thailand' 'Timor-Leste' 'Togo'
 'Tokelau' 'Tonga' 'Trinidad and Tobago' 'Tunisia' 'Turkey' 'Turkmen
istan'
 'Turks and Caicos Islands' 'Tuvalu' 'Uganda' 'Ukraine'
 'United Arab Emirates' 'United Kingdom' 'United Republic of Tanzani
a'
 'United States of America' 'United States Virgin Islands' 'Uruguay'
 'Uzbekistan' 'Vanuatu' 'Venezuela (Bolivarian Republic of)' 'Viet N
am'
 'Wallis and Futuna Islands' 'Western Sahara' 'Yemen' 'Zambia' 'Zimb
abwe']
```

Line plot

Line diagrams works best with a series of data, assuming continuous change between the known discrete values.
Let's visualize the total and male population of *Hungary* between the years 1950 an 2019.

First filter the rows based on the country for *Hungary* and set the year as the index column.

In [26]:

```
hungary = population_history[population_history['Country'] == 'Hungary']
hungary.set_index('Year', drop=False, inplace=True)
display(hungary)
```

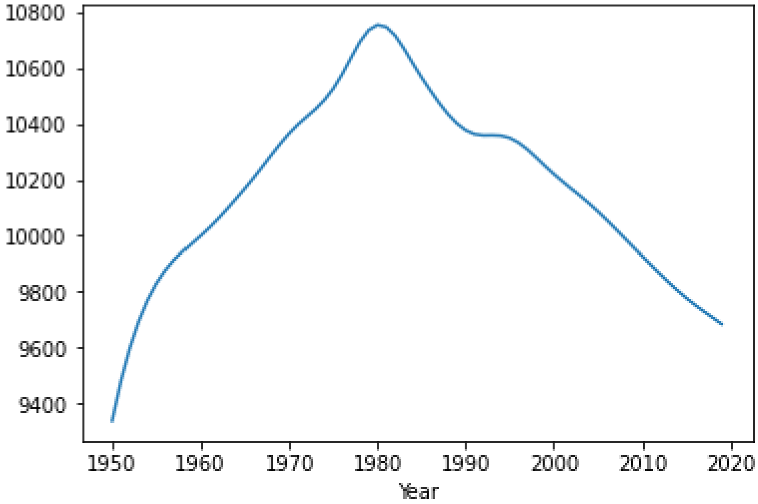
	Country	Year	PopMale	PopFemale	PopTotal	PopDensity
Year						
1950	Hungary	1950	4494.406	4843.312	9337.718	103.145
1951	Hungary	1951	4573.710	4906.897	9480.607	104.723
1952	Hungary	1952	4637.570	4960.372	9597.942	106.019
1953	Hungary	1953	4687.602	5005.300	9692.902	107.068
1954	Hungary	1954	4725.599	5043.080	9768.679	107.905
...	...	...	...	...	...	...
2015	Hungary	2015	4646.716	5131.209	9777.925	108.008
2016	Hungary	2016	4636.375	5116.595	9752.970	107.732
2017	Hungary	2017	4626.816	5103.006	9729.822	107.476
2018	Hungary	2018	4617.623	5089.879	9707.502	107.230
2019	Hungary	2019	4608.250	5076.430	9684.680	106.978

70 rows × 6 columns

Now a line plot on the total population change of Hungary between 1950 and 2019 can be displayed.

In [27]:

```
hungary['PopTotal'].plot(kind='line')
plt.show()

# same:
#hungary.plot(kind='line', x='Year', y='PopTotal')
#plt.show()
```



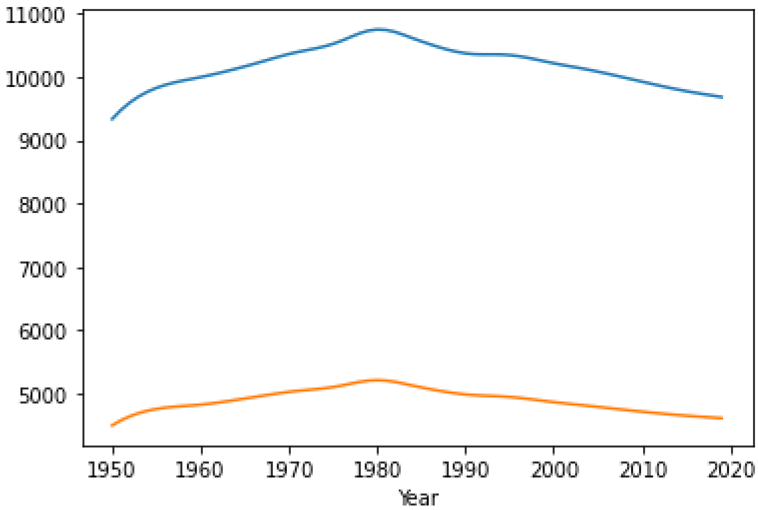
Multiple column diagrams

Let's use multiple columns in the previous line plot, and add the male population to the diagram as a second line.

Multiple plot data can be generated with the `plot()` method of Pandas *Series*. Calling the `show()` function of the `matplotlib.pyplot` module will visualize them on a single diagram.

In [28]:

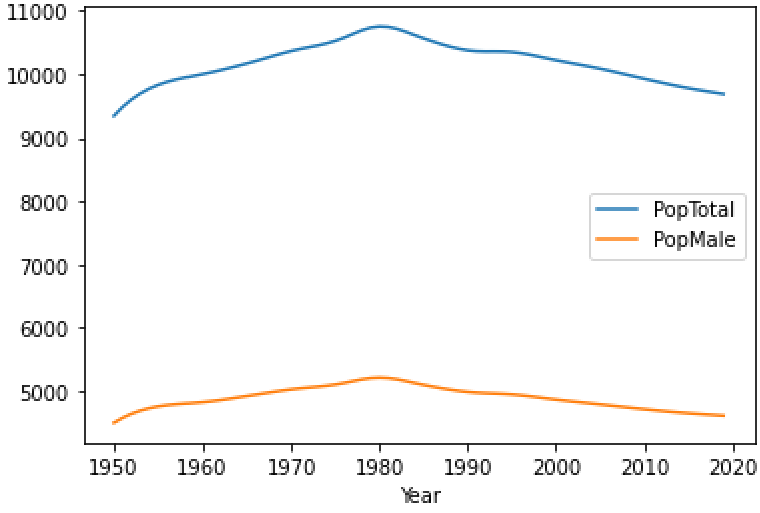
```
hungary['PopTotal'].plot(kind='line')
hungary['PopMale'].plot(kind='line')
plt.show()
```



Add legend to the diagram:

In [29]:

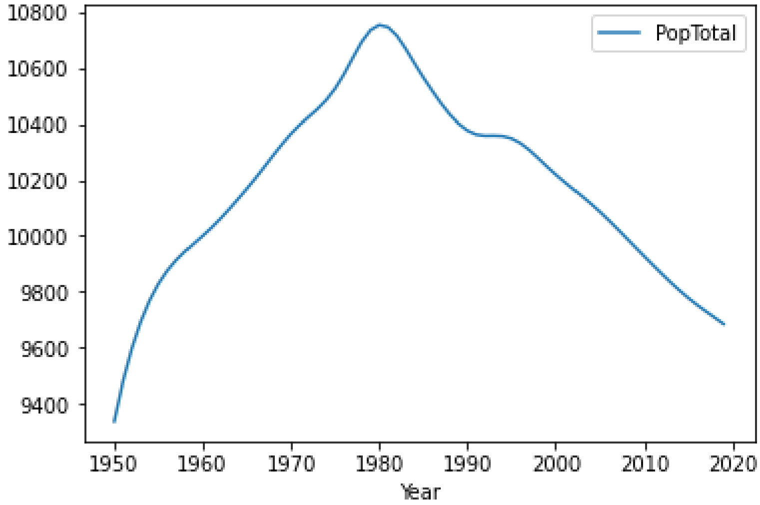
```
hungary['PopTotal'].plot(kind='line', legend=True)
hungary['PopMale'].plot(kind='line', legend=True)
plt.show()
```

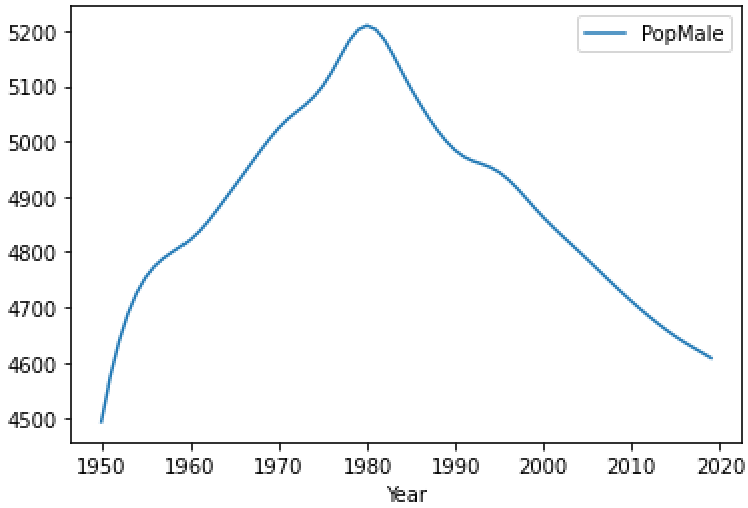


The same can be done by calling the `plot()` method of a *Pandas DataFrame*. Be aware though, that in this case each plot will be displayed in an individual diagram:

In [30]:

```
hungary.plot(kind='line', x='Year', y='PopTotal', legend=True)
hungary.plot(kind='line', x='Year', y='PopMale', legend=True)
plt.show()
```

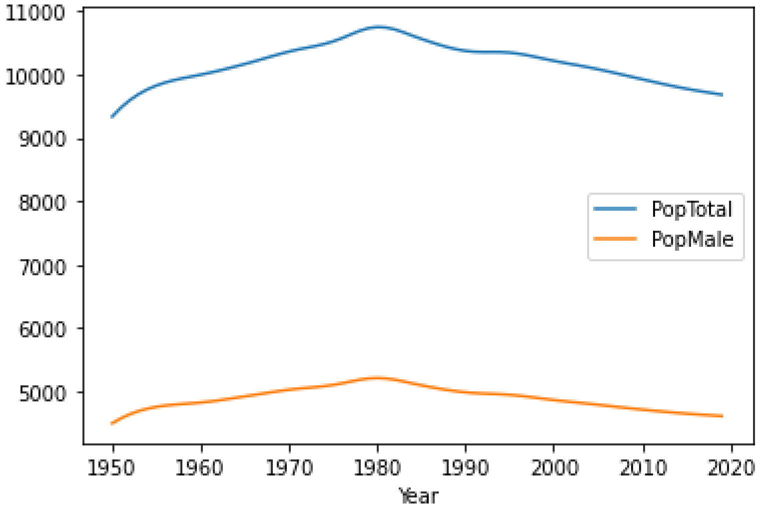




This can be fixed by explicitly configuring matplotlib to use the same *axis object* for visualization for both diagrams:

In [31]:

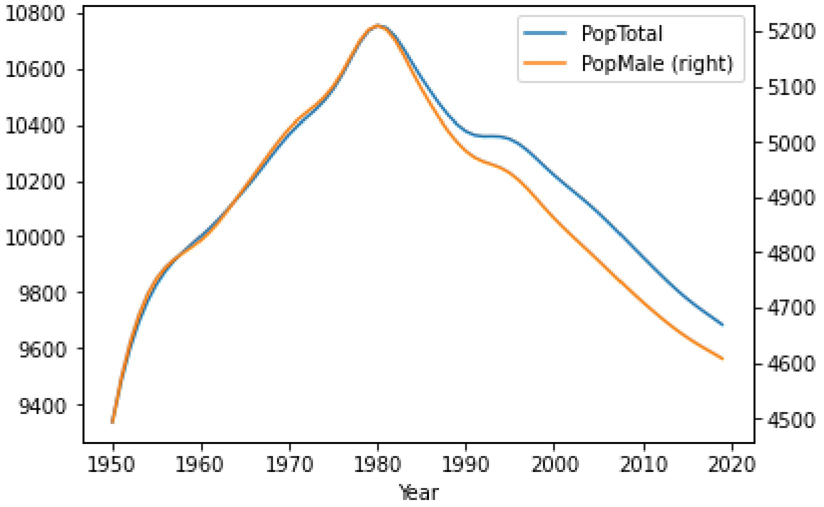
```
ca = plt.gca() # gca = get current axis configuration object
hungary.plot(kind='line', x='Year', y='PopTotal', ax=ca, legend=True) # use the
 ca axis configuration object
hungary.plot(kind='line', x='Year', y='PopMale', ax=ca, legend=True) # use the c
a axis configuration object
plt.show()
```



Use a different, secondary scale for the male population.

In [32]:

```
hungary['PopTotal'].plot(kind='line', legend=True)
hungary['PopMale'].plot(kind='line', secondary_y=True, legend=True)
plt.show()
```



Data grouping

Pandas supports the grouping of data by the given column(s), which then can be used also for visualization.

Select 10 countries by your choice.

In [33]:

```
selected_countries = pd.Series(['Hungary', 'Germany', 'France', 'United Kingdom'
, 'Romania', 'Oman', 'Libya', 'Turkey', 'Chile', 'Viet Nam'])
display(selected_countries)
```

```
0          Hungary
1          Germany
2           France
3    United Kingdom
4          Romania
5             Oman
6            Libya
7           Turkey
8            Chile
9         Viet Nam
dtype: object
```

Select the rows of the original DataFrame for these selected countries.

In [34]:

```
selected_history = population_history[population_history['Country'].isin(selecte
d_countries)]
display(selected_history)
```

	Country	Year	PopMale	PopFemale	PopTotal	PopDensity
3150	Chile	1950	3335.670	3262.848	6598.518	8.875
3151	Chile	1951	3398.318	3331.262	6729.580	9.051
3152	Chile	1952	3465.497	3404.217	6869.714	9.239
3153	Chile	1953	3535.877	3480.588	7016.465	9.437
3154	Chile	1954	3608.433	3559.476	7167.909	9.640
...	...	...	...	...	...	...
16375	Viet Nam	2015	46197.466	46479.616	92677.082	298.891
16376	Viet Nam	2016	46696.272	46944.163	93640.435	301.998
16377	Viet Nam	2017	47193.015	47407.628	94600.643	305.094
16378	Viet Nam	2018	47680.864	47865.095	95545.959	308.143
16379	Viet Nam	2019	48151.352	48310.756	96462.108	311.098

700 rows × 6 columns

The `selected_history` *DataFrame* now contains all historical data for the selected 10 countries.

Visualize the population change of the selected 10 countries for the time period 1950-2019 in a line diagram. To achieve this, we first group the `selected_history` *DataFrame* by the `Country` *Series*:

In [35]:

```
selected_history.groupby('Country')
```

Out[35]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f148327a2
80>
```

We have got a `DataFrameGroupBy` object, which can be converted to a list:

In [36]:

```
grouped_history = list(selected_history.groupby('Country'))
print("Length: {0}".format(len(grouped_history)))
```

```
Length: 10
```

Each item of the list contains all records for a given *country* (the column used for groupping):

In [37]:

```
print(grouped_history[0])
```

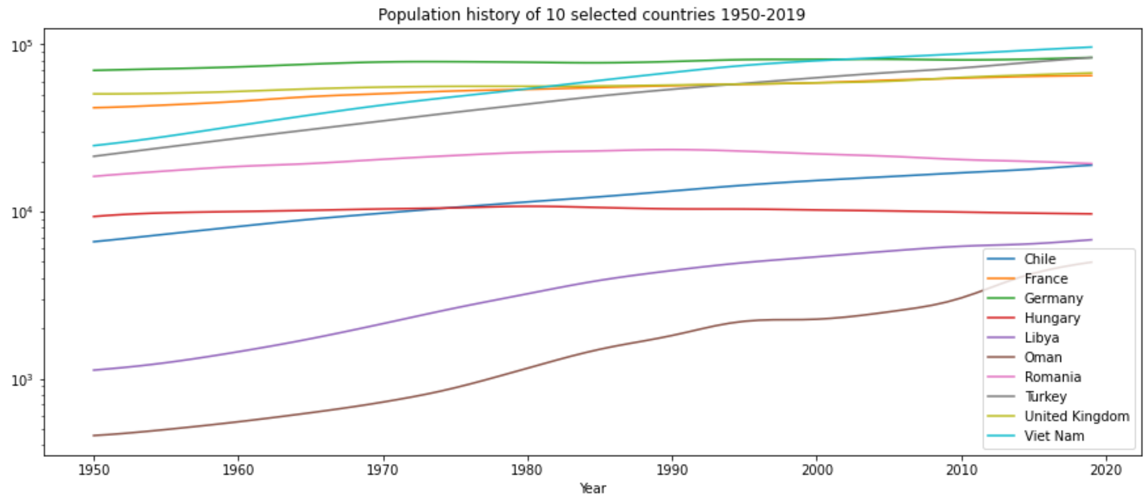
```
('Chile',      Country  Year   PopMale  PopFemale   PopTotal  PopDen
sity
3150   Chile  1950  3335.670   3262.848   6598.518      8.875
3151   Chile  1951  3398.318   3331.262   6729.580      9.051
3152   Chile  1952  3465.497   3404.217   6869.714      9.239
3153   Chile  1953  3535.877   3480.588   7016.465      9.437
3154   Chile  1954  3608.433   3559.476   7167.909      9.640
...      ...   ...       ...        ...        ...        ...
3215   Chile  2015  8844.800   9124.556  17969.356     24.168
3216   Chile  2016  8965.258   9243.814  18209.072     24.490
3217   Chile  2017  9097.252   9373.183  18470.435     24.841
3218   Chile  2018  9228.416   9500.750  18729.166     25.189
3219   Chile  2019  9341.774   9610.261  18952.035     25.489

[70 rows x 6 columns])
```

Question: what happens if we group by the year column?

Based on the grouped *DataFrame*, we select the `PopTotal` *Series* and create a line plot. First the `Year` column is set as an index to be used for the X axis.

In [38]:

```
selected_history.set_index('Year', inplace=True, drop=False)
selected_history.groupby('Country')['PopTotal'].plot(
    kind='line', logy=True,
    figsize=[15, 6], legend=True,
    title='Population history of 10 selected countries 1950-2019')
plt.show()
```



Aggregate functions

Aggregate functions (`min` , `max` , `mean` , `median` , `sum` , etc.) transforms a group of values to a single value. By calling on aggregate function on a grouped *DataFrame*, the aggregated value of each group is calculated.

Let's calculate the largest population for each country they ever had between 1950 and 2019.

In [39]:

```
population_history.groupby('Country').max()
```

Out[39]:

Country	Year	PopMale	PopFemale	PopTotal	PopDensity
Afghanistan	2019	19529.727	18512.030	38041.757	58.269
Albania	2019	1682.757	1611.474	3286.070	119.930
Algeria	2019	21749.666	21303.388	43053.054	18.076
American Samoa	2019	NaN	NaN	59.684	298.420
Andean Community	2019	55331.532	56405.132	111736.664	30.027
...	...	...	...	...	...
Wallis and Futuna Islands	2019	NaN	NaN	15.098	107.843
Western Sahara	2019	304.755	277.703	582.458	2.190
Yemen	2019	14692.284	14469.638	29161.922	55.234
Zambia	2019	8843.214	9017.820	17861.034	24.026
Zimbabwe	2019	6983.353	7662.120	14645.473	37.858

239 rows × 5 columns

Sort the result by the `PopTotal` and only display the `PopTotal` :

In [40]:

```
largest_pop = population_history.groupby('Country').max().sort_values(by = 'PopTotal')['PopTotal']
display(largest_pop)
```

```
Country
Holy See                             0.909
Tokelau                              1.953
Falkland Islands (Malvinas)          3.372
Niue                                 5.242
Saint Pierre and Miquelon            6.435
                                  ...
Pakistan                        216565.317
Indonesia                       270625.567
United States of America        329064.917
India                          1366417.756
China                          1433783.692
Name: PopTotal, Length: 239, dtype: float64
```

Summary exercises on plotting

Exercise 1

Task: Use the *World Countries dataset* defined in the `countries` variable. That dataset contained the *region* for each country. Compute for each region how many countries belong to them. Visualize the results in a pie a chart.

Hint: use groupping.

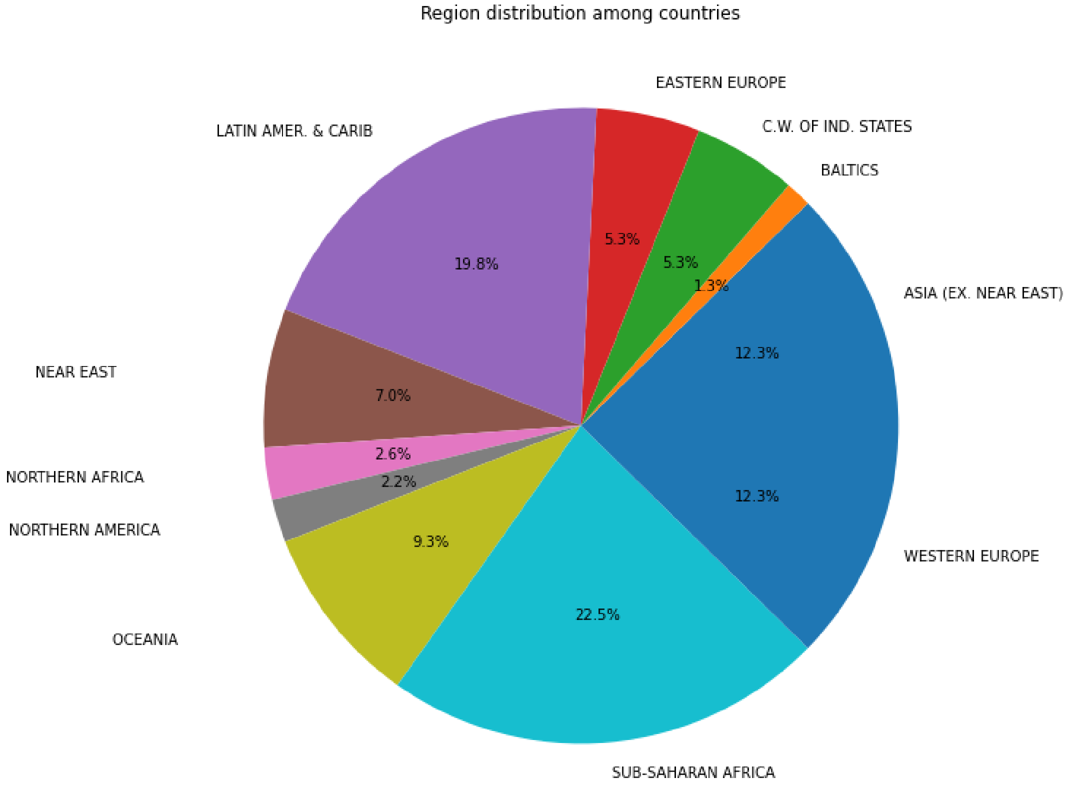
In [41]:

```
countries_by_region = countries.groupby('region').count()['country']
display(countries_by_region)
```

```
region
ASIA (EX. NEAR EAST)                   28
BALTICS                                 3
C.W. OF IND. STATES                    12
EASTERN EUROPE                         12
LATIN AMER. & CARIB                    45
NEAR EAST                              16
NORTHERN AFRICA                         6
NORTHERN AMERICA                        5
OCEANIA                                21
SUB-SAHARAN AFRICA                     51
WESTERN EUROPE                         28
Name: country, dtype: int64
```

In [42]:

```
countries_by_region.plot(kind='pie', figsize=[10,10], autopct='%.1f%%', label=""
,
                           title="Region distribution among countries")
plt.show()
```



Exercise 2

Task: Calculate the accumulated population of the world for each year between 1950 and 2019 based on the *Population History dataset* stored in the `population_history` variable.
Create a bar diagram visualizing how the aggregated population changed over the years.

In [43]:

```
aggregated_by_year = population_history.groupby('Year').sum()
display(aggregated_by_year)
```

	PopMale	PopFemale	PopTotal	PopDensity
Year				
1950	1278875.631	1282748.044	2562089.503	42672.546
1951	1303179.841	1306685.514	2610335.875	42728.190
1952	1327130.022	1330216.610	2657822.039	42972.585
1953	1351073.239	1353698.802	2705252.614	43345.374
1954	1375294.431	1377424.036	2753204.644	43831.847
...	...	...	...	...
2015	3764759.824	3703476.149	7469342.451	106178.776
2016	3807875.525	3745887.267	7554873.938	107422.436
2017	3850938.612	3788132.946	7640187.980	108633.385
2018	3893745.012	3830090.171	7724957.236	109803.197
2019	3936030.563	3871616.311	7808774.650	110916.555

70 rows × 4 columns

In [44]:

```
aggregated_by_year.plot(kind='bar', y='PopTotal', figsize=[15, 4], width=0.8, co
lor='orange')
plt.show()
```