05_pandas

November 9, 2022

1 pandas

`pandas` is an open source library that brings high-performance, easy-to-use data structures and analysis tools to the Python programming language. It's core features are its 1-dimensional `Series` and 2-dimensional `DataFrame` objects. The former is like a NumPy array (`pandas` is built on top of NumPy) with an explicit index, and the latter is a bit like an Excel worksheet.

The 10 minutes to pandas and getting started guides are excellent places to learn more, but read on for my own tour of the library which draws comparisons between pandas and familiar spreadsheet programs (e.g., Microsoft Excel, Apple Numbers, LibreOffice Calc, Google Sheets).

1.1 Importing pandas

The community agreed custom for importing `pandas` is:

```
import pandas as pd
```

This gives us access to all `pandas` functionality, which means it is a bit like clicking on the desktop icon that launches a spreadsheet application. Let's import `pandas`, and while we are at it, let's also import `numpy` and `matplotlib` (and set some configuration options).

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
```

We'll begin with a detailed look at the `Series` and `DataFrame` objects.

1.2 Series

`pd.Series` is much like a 1-dimensional NumPy array, but with explicit axis labels, the ability to hold any data type, and various feature and method enhancements to simplify working with complex data. We can think of a `Series` as the rough equivalent of a column in a spreadsheet application.

The basic way of making a `Series` is as follows:

```
s = pd.Series(data, index=index)
```

Note that the argument for `data` can take various forms, such as a Python dictionary, a `numpy` array, a list, or even a single value. If provided, `index` must be a list of axis labels with the same length as `data`.

Here's a simple `Series` of random numbers.

[2]:
```
s = pd.Series(np.random.random(100))
print(s)
```

```
0     0.110035
1     0.398505
2     0.783878
3     0.776024
4     0.343782
        …
95    0.153074
96    0.580834
97    0.695021
98    0.302373
99    0.391739
Length: 100, dtype: float64
```

We didn't specify an `index`, so a default integer-based index (starting at 0) was applied, which we can see to the left of the actual values. Below the index and values we can also see the length and data type.

Now, let's make a `Series` containing the letters of the alphabet and assign also an `index` and a `name`.

[3]:
```
letters = list('abcdefghijklmnopqrstuvwxyz')
s = pd.Series(letters, index=range(1, 27), name='lowercase')
print(s)
```

```
1     a
2     b
3     c
4     d
5     e
6     f
7     g
8     h
9     i
10    j
11    k
12    l
13    m
14    n
15    o
16    p
17    q
```

```
18    r
19    s
20    t
21    u
22    v
23    w
24    x
25    y
26    z
Name: lowercase, dtype: object
```

A few things to note here. First, the `index` doesn't default to zero because we specified our own labels. Second, the `name` we provided is shown below the values (this is analogous to a column header in spreadsheet applications). Third, the data type is `object`, denoting that the `Series` contains `str` data.

1.3 DataFrame

A `DataFrame` is a 2-dimensional data structure consisting of rows and columns, which makes it similar to a worksheet in Microsfot Excel. Each column in a `DataFrame` is a `Series`, which by now we know to be a feature-enhanced, explicitly indexed, NumPy array that can contain any type of data. A `DataFrame`, then, is a collection of `Series` that share a common `Index`.

An easy way to create a basic `DataFrame`is to pass a Python `dict` to the `pd.DataFrame(...)` constructor. In this scenario, the keys will serve as names for the columns, and the values, which must be convertible to a series-like data structure, will be the data.

[4]:
```
# It's the familiar shopping example again...
df = pd.DataFrame(
    {
        'item': ['apples', 'bananas', 'bread', 'yoghurt', 'pasta', 'coffee',␣
↪'butter', 'carrots'],
        'unit_price': np.array([2.20, 1.85, 1.10, 1.20, 1.45, 2.80, 2.40, .89],␣
↪dtype='float32'),
        'quantity': np.array([2, 1, 2, 4, 2, 1, 1, 1]),
        'date_of_purchase': pd.Timestamp('20221208'),
        'shop': pd.Categorical(['Aldi'] * 8),
        'paid_cash': True
    }
)
df
```

[4]:
```
      item  unit_price  quantity date_of_purchase  shop  paid_cash
0   apples        2.20         2       2022-12-08  Aldi       True
1  bananas        1.85         1       2022-12-08  Aldi       True
2    bread        1.10         2       2022-12-08  Aldi       True
3  yoghurt        1.20         4       2022-12-08  Aldi       True
4    pasta        1.45         2       2022-12-08  Aldi       True
5   coffee        2.80         1       2022-12-08  Aldi       True
```

```
6   butter         2.40          1        2022-12-08  Aldi        True
7  carrots         0.89          1        2022-12-08  Aldi        True
```

As with NumPy arrays, the `DataFrame` object stores some basic information as attributes. Below, we get information about shape, number of dimensions, size (number of elements), and the data types for each column of `df`.

[5]:
```
print('Shape:                ', df.shape)
print('Number of dimensions: ', df.ndim)
print('Number of elements:   ', df.size)
print('Data types:\n')
print(df.dtypes)
```

```
Shape:                 (8, 6)
Number of dimensions:  2
Number of elements:    48
Data types:

item                        object
unit_price                 float32
quantity                     int64
date_of_purchase    datetime64[ns]
shop                      category
paid_cash                     bool
dtype: object
```

We can also see how much memory (in bytes) is being used by each column.

[6]:
```
df.memory_usage()
```

[6]:
```
Index               128
item                 64
unit_price           32
quantity             64
date_of_purchase     64
shop                124
paid_cash             8
dtype: int64
```

The actual values stored in a `DataFrame` are accessible via the `.values` attribute. Inspection of these values shows that `pandas` is built right on top of NumPy.

[7]:
```
print("The values of the data frame are: ")
print(df.values)
print("The type of the values is ", type(df.values))
```

```
The values of the data frame are:
[['apples' 2.200000047683716 2 Timestamp('2022-12-08 00:00:00') 'Aldi'
  True]
```

```
 ['bananas' 1.850000023841858 1 Timestamp('2022-12-08 00:00:00') 'Aldi'
  True]
 ['bread' 1.100000023841858 2 Timestamp('2022-12-08 00:00:00') 'Aldi'
  True]
 ['yoghurt' 1.2000000476837158 4 Timestamp('2022-12-08 00:00:00') 'Aldi'
  True]
 ['pasta' 1.4500000476837158 2 Timestamp('2022-12-08 00:00:00') 'Aldi'
  True]
 ['coffee' 2.799999952316284 1 Timestamp('2022-12-08 00:00:00') 'Aldi'
  True]
 ['butter' 2.4000000953674316 1 Timestamp('2022-12-08 00:00:00') 'Aldi'
  True]
 ['carrots' 0.8899999856948853 1 Timestamp('2022-12-08 00:00:00') 'Aldi'
  True]]
The type of the values is  <class 'numpy.ndarray'>
```

1.4 Viewing, selecting and indexing

If you are new to `pandas`, it may sometimes feel like the data are hidden away and out of reach, especially in comparison to spreadsheet environments where data are generally always on the screen and can be selected and browsed by clicking and scrolling the mouse. Thankfully, `pandas` has some handy tools for viewing a `DataFrame`.

`DataFrame.head()` can be used to view the top of a frame, and `DataFrame.tail()` to view the bottom. The default number of rows to display is `5`, but we can change this by specifying a number.

[8]:
```
# Show the first three rows of df
df.head(3)
```

[8]:

	item	unit_price	quantity	date_of_purchase	shop	paid_cash
0	apples	2.20	2	2022-12-08	Aldi	True
1	bananas	1.85	1	2022-12-08	Aldi	True
2	bread	1.10	2	2022-12-08	Aldi	True

[9]:
```
# Show the last three rows of df
df.tail(3)
```

[9]:

	item	unit_price	quantity	date_of_purchase	shop	paid_cash
5	coffee	2.80	1	2022-12-08	Aldi	True
6	butter	2.40	1	2022-12-08	Aldi	True
7	carrots	0.89	1	2022-12-08	Aldi	True

The row and column indices themselves can also be accessed and operated upon (note that we didn't specify a row index at the time of creation, so we got a default `RangeIndex`).

[10]:
```
# View the row index of df
df.index
```

[10]:
```
RangeIndex(start=0, stop=8, step=1)
```

[11]:
```
# View the column index of df
df.columns
```

[11]:
```
Index(['item', 'unit_price', 'quantity', 'date_of_purchase', 'shop',
       'paid_cash'],
      dtype='object')
```

Let's talk about selecting columns. To select a single column of a `DataFrame`, put its name in square brackets. This is a bit like clicking on the column header in a spreadsheet to highlight the entire column. The result of this operation is a `Series`.

[12]:
```
df['unit_price']
```

[12]:
```
0    2.20
1    1.85
2    1.10
3    1.20
4    1.45
5    2.80
6    2.40
7    0.89
Name: unit_price, dtype: float32
```

As long as the column name doesn't contain whitespace or special characters, single columns may also be accessed in equivalent fashion via `.` notation. I often find this more convenient, as it's easier to type.

[13]:
```
df.unit_price
```

[13]:
```
0    2.20
1    1.85
2    1.10
3    1.20
4    1.45
5    2.80
6    2.40
7    0.89
Name: unit_price, dtype: float32
```

Selecting multiple columns at the same time requires a list of column names, and returns a `DataFrame` (this is like doing `Ctrl+click` or `Ctrl+drag` to select multiple columns in a spreadsheet).

[14]:
```
df[['item', 'quantity', 'unit_price']]
```

[14]:
```
      item  quantity  unit_price
0   apples         2        2.20
1  bananas         1        1.85
2    bread         2        1.10
```

```
3  yoghurt         4        1.20
4    pasta         2        1.45
5   coffee         1        2.80
6   butter         1        2.40
7  carrots         1        0.89
```

[] can be used to slice by row

[15]:
```python
df[2:5]
```

[15]:
```
      item  unit_price  quantity date_of_purchase  shop  paid_cash
2    bread        1.10         2       2022-12-08  Aldi       True
3  yoghurt        1.20         4       2022-12-08  Aldi       True
4    pasta        1.45         2       2022-12-08  Aldi       True
```

`pandas` also supports **label-based indexing**, which is extremely useful for selecting individual values or cross-sections of a `DataFrame`.

Label-based indexing is performed using the `.loc[]` and `.at[]` access methods. Note these are square, and not round, brackets.

[16]:
```python
# Select rows 4:7 and the specified columns
df.loc[4:7, ['item', 'unit_price', 'quantity']]
```

[16]:
```
      item  unit_price  quantity
4    pasta        1.45         2
5   coffee        2.80         1
6   butter        2.40         1
7  carrots        0.89         1
```

[17]:
```python
# Select the value in row=3, column='item'
df.at[3, 'item']
```

[17]: 'yoghurt'

Positional indexing can be achieved with the `.iloc[]` and `.iat[]` access methods. The difference here is that we ***must*** provide integers (starting at `0`). The behavior is very similar to array slicing in Python and NumPy.

[18]:
```python
# Select the 6th row
df.iloc[5]
```

[18]:
```
item                              coffee
unit_price                           2.8
quantity                               1
date_of_purchase     2022-12-08 00:00:00
shop                                Aldi
paid_cash                           True
Name: 5, dtype: object
```

[19]:
```python
# Select the value in the second row of the first column
df.iat[1, 0]
```

[19]:
```
'bananas'
```

Boolean indexing is another common way to select a subset of data from a frame. Suppose we cared only about rows where the `unit_price` was over `£2.00`.

[20]:
```python
df[df['unit_price'] > 2.]
```

[20]:

	item	unit_price	quantity	date_of_purchase	shop	paid_cash
0	apples	2.2	2	2022-12-08	Aldi	True
5	coffee	2.8	1	2022-12-08	Aldi	True
6	butter	2.4	1	2022-12-08	Aldi	True

This works, because the expression inside the square brackets evaluates as a `Series` of `bool`, and the indexing operation ensures that we only get the rows where the expression comes up `True`.

[21]:
```python
df['unit_price']>2.
```

[21]:
```
0     True
1    False
2    False
3    False
4    False
5     True
6     True
7    False
Name: unit_price, dtype: bool
```

We can get quite fancy by chaining multiple expressions together. For example, if we only wanted rows where the `unit_price` is greater than `£1.00` and less than `£2.00`.

[22]:
```python
df[((df['unit_price'] > 1.) & (df['unit_price'] < 2.))]
```

[22]:

	item	unit_price	quantity	date_of_purchase	shop	paid_cash
1	bananas	1.85	1	2022-12-08	Aldi	True
2	bread	1.10	2	2022-12-08	Aldi	True
3	yoghurt	1.20	4	2022-12-08	Aldi	True
4	pasta	1.45	2	2022-12-08	Aldi	True

1.5 World population dataset

The best way to learn `pandas` is to use it with real data, so let's explore some more advanced features with an actual dataset.

In the `data` folder of the course materials, there's a file called `world_population.csv` which I sourced from an excellent data science website called Kaggle. The file contains historical population data for every country/territory in the world, along with various other parameters. Here's what's in the file:

Variable	Definition
Rank	Rank by Population
CCA3	3 Digit Country/Territories Code
Country	Name of the Country/Territories
Capital	Name of the Capital
Continent	Name of the Continent
2022 Population	Population of the Country/Territories in the year 2022
2020 Population	Population of the Country/Territories in the year 2020
2015 Population	Population of the Country/Territories in the year 2015
2010 Population	Population of the Country/Territories in the year 2010
2000 Population	Population of the Country/Territories in the year 2000
1990 Population	Population of the Country/Territories in the year 1990
1980 Population	Population of the Country/Territories in the year 1980
1970 Population	Population of the Country/Territories in the year 1970
Area (km²)	Area size of the Country/Territories in square kilometer
Density (per km²)	Population Density per square kilometer
Growth Rate	Population Growth Rate by Country/Territories
World Population Percentage	The population percentage by each Country/Territories

Let's start by loading and inspecting the data.

Because its a `CSV` file, `pd.read_csv(...)` is right tool for the job.

[23]:
```python
df = pd.read_csv('../data/world_population.csv')
df
```

[23]:
```
     Rank CCA3            Country           Capital Continent  \
0      36  AFG        Afghanistan             Kabul      Asia
1     138  ALB            Albania            Tirana    Europe
2      34  DZA            Algeria           Algiers    Africa
3     213  ASM     American Samoa         Pago Pago   Oceania
4     203  AND            Andorra  Andorra la Vella    Europe
..    …   …                  …                …        …
229   226  WLF  Wallis and Futuna          Mata-Utu   Oceania
230   172  ESH     Western Sahara          El Aaiún    Africa
231    46  YEM              Yemen             Sanaa      Asia
232    63  ZMB             Zambia            Lusaka    Africa
233    74  ZWE           Zimbabwe            Harare    Africa

     2022 Population  2020 Population  2015 Population  2010 Population  \
0           41128771         38972230         33753499         28189672
1            2842321          2866849          2882481          2913399
2           44903225         43451666         39543154         35856344
3              44273            46189            51368            54849
```

```
4             79824          77700          71746          71519
..              …              …              …              …
229           11572          11655          12182          13142
230          575986         556048         491824         413296
231        33696614       32284046       28516545       24743946
232        20017675       18927715       16248230       13792086
233        16320537       15669666       14154937       12839771

     2000 Population  1990 Population  1980 Population  1970 Population  \
0           19542982         10694796         12486631         10752971
1            3182021          3295066          2941651          2324731
2           30774621         25518074         18739378         13795915
3              58230            47818            32886            27075
4              66097            53569            35611            19860
..               …                …                …                …
229            14723            13454            11315             9377
230           270375           178529           116775            76371
231         18628700         13375121          9204938          6843607
232          9891136          7686401          5720438          4281671
233         11834676         10113893          7049926          5202918

     Area (km²)  Density (per km²)  Growth Rate  World Population Percentage
0        652230            63.0587       1.0257                         0.52
1         28748            98.8702       0.9957                         0.04
2       2381741            18.8531       1.0164                         0.56
3           199           222.4774       0.9831                         0.00
4           468           170.5641       1.0100                         0.00
..          …                  …          …                          …
229         142            81.4930       0.9953                         0.00
230      266000             2.1654       1.0184                         0.01
231      527968            63.8232       1.0217                         0.42
232      752612            26.5976       1.0280                         0.25
233      390757            41.7665       1.0204                         0.20

[234 rows x 17 columns]
```

We now have a `DataFrame` with 17 columns and 234 rows, which appear to be sorted by `Country` in ascending alphabetical order. All in all, these are clean and well organized data, but if we sorted the rows by `Rank`, we could get a quick and easy insight into the most and least populous countries. We can do this using `.sort_values(...)`.

[24]:
```python
# Sort the rows by Rank in descending order
df.sort_values('Rank', ascending=True)
```

[24]:
```
     Rank CCA3          Country         Capital      Continent  \
41      1  CHN            China         Beijing           Asia
92      2  IND            India       New Delhi           Asia
```

221	3	USA	United States	Washington, D.C.	North America
93	4	IDN	Indonesia	Jakarta	Asia
156	5	PAK	Pakistan	Islamabad	Asia
..	…	…	…	…	…
137	230	MSR	Montserrat	Brades	North America
64	231	FLK	Falkland Islands	Stanley	South America
150	232	NIU	Niue	Alofi	Oceania
209	233	TKL	Tokelau	Nukunonu	Oceania
226	234	VAT	Vatican City	Vatican City	Europe

	2022 Population	2020 Population	2015 Population	2010 Population	\
41	1425887337	1424929781	1393715448	1348191368	
92	1417173173	1396387127	1322866505	1240613620	
221	338289857	335942003	324607776	311182845	
93	275501339	271857970	259091970	244016173	
156	235824862	227196741	210969298	194454498	
..	…	…	…	…	
137	4390	4500	5059	4938	
64	3780	3747	3408	3187	
150	1934	1942	1847	1812	
209	1871	1827	1454	1367	
226	510	520	564	596	

	2000 Population	1990 Population	1980 Population	1970 Population	\
41	1264099069	1153704252	982372466	822534450	
92	1059633675	870452165	696828385	557501301	
221	282398554	248083732	223140018	200328340	
93	214072421	182159874	148177096	115228394	
156	154369924	115414069	80624057	59290872	
..	…	…	…	…	
137	5138	10805	11452	11402	
64	3080	2332	2240	2274	
150	2074	2533	3637	5185	
209	1666	1669	1647	1714	
226	651	700	733	752	

	Area (km^2)	Density (per km^2)	Growth Rate	World Population Percentage
41	9706961	146.8933	1.0000	17.88
92	3287590	431.0675	1.0068	17.77
221	9372610	36.0935	1.0038	4.24
93	1904569	144.6529	1.0064	3.45
156	881912	267.4018	1.0191	2.96
..	…	…	…	…
137	102	43.0392	0.9939	0.00
64	12173	0.3105	1.0043	0.00
150	260	7.4385	0.9985	0.00
209	12	155.9167	1.0119	0.00

```
226            1            510.0000       0.9980                              0.00

[234 rows x 17 columns]
```

This is great, but everyone knows that China is the most populated country in the world. Surely there is a more nuanced story to be told. Let's start by calculating the most recent estimate of the global population by summing all of the values in the `2022 Population` column.

[25]:
```
print('The global population in 2022 is: ')
df['2022 Population'].sum()
```

```
The global population in 2022 is:
```

[25]:
```
7973413042
```

That is believable. But now I'm curious how that is distributed by `Continent`.

A `.groupby` operation is perfect for this job.

> Read more about `.groupby()`

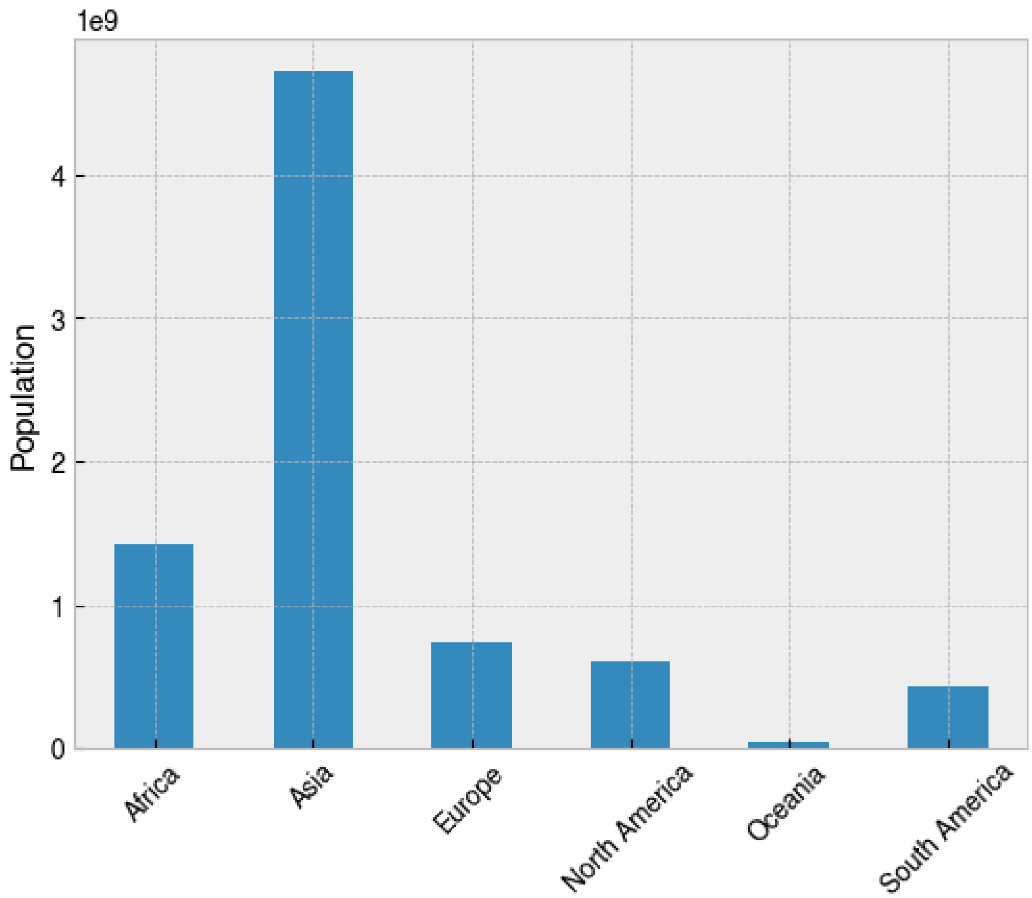
[26]:
```
# Group by Continent and sum the values
df.groupby('Continent')['2022 Population'].sum()
```

[26]:
```
Continent
Africa           1426730932
Asia             4721383274
Europe            743147538
North America     600296136
Oceania            45038554
South America     436816608
Name: 2022 Population, dtype: int64
```

Unsurprisingly, Asia is the most highly populated continent. A bar chart may help us to appreciate this more fully. We can easily make a bar chart using the `.plot()` method for `Series` and `DataFrames`.

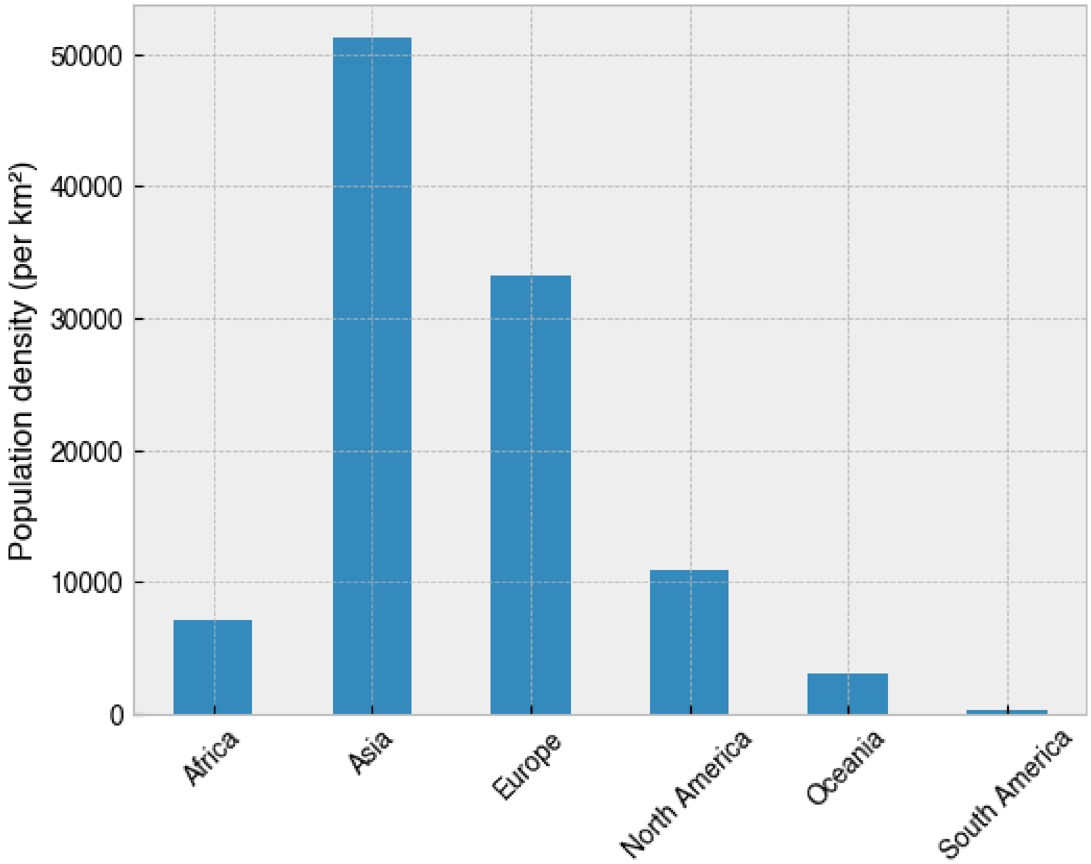
> Read more about `.plot()`

[27]:
```
# Repeat the above operation, but this time chain a plotting method to the end
(
 df.groupby('Continent')['2022 Population']
 .sum()
 .plot(kind='bar', rot=45, xlabel='', ylabel='Population')
);
```



So Asia is the most populated continent, but surely that's just because its the biggest. How does the population relate to the total size of the continent? The `Density (per km²)` column looks like it was calculated by dividing the population of a country by its total area, so repeating the operation with this variable will help to develop the story.

[28]:
```
# As above, but with the 'Density (per km²)' column instead
(
 df.groupby('Continent')['Density (per km²)']
 .sum()
 .plot(kind='bar', rot=45, xlabel='', ylabel='Population density (per km²)')
);
```



This paints a different picture. Most striking is the difference for Europe. Though it is the third most populated continent, Europe comes second for population density, and by a long way in comparison to the others. This is especially true in comparison with South America, whose population is very sparse indeed.

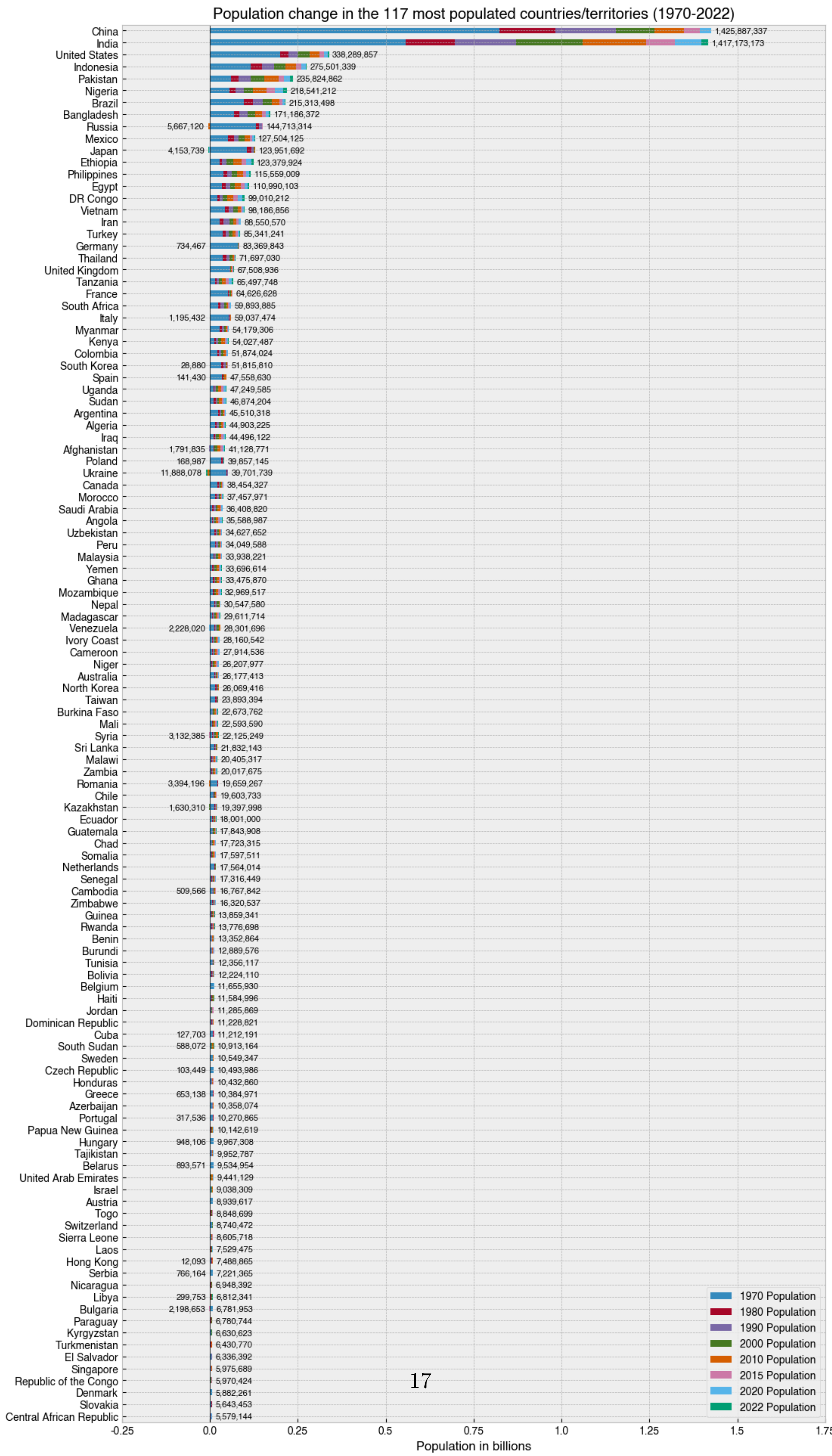
1.6 The complete picture - population growth

To tell the overall story of these population data, I came up with my own two-figure solution, which I feel does a pretty good job. I sort the data by `Rank`, calculate a column-wise differential on the `<date> Population` columns, and then plot the result for each country in a stacked horizontal bar chart, with bars to the left show population decline, and bars to right show population growth for that period. To avoid ambiguity, I put the total population values to the side of each bar. On the right, these values indicate the population for that country in 2022, and on the left, they indicate the total population decline since 1970.

[33]:
```
# Get the columns with the population data
pop_cols = df.columns[df.columns.str.endswith('Population')]

# Sort and select data
data = (df
```

```
    .set_index('Country')  # Set 'Country' as the index
    .sort_values('Rank', ascending=True)[pop_cols]  # Sort by 'Rank' and keep only␣
↪the columns with population data
    .iloc[0:117]  # Choose 117 MOST populated countries with location-based␣
↪indexing
    .loc[:, lambda df_: reversed(df_.columns)]  # Reverse the order of the columns␣
↪in a fancy way
)

# Pull out the base population at 1970
base_pop_1970 = data['1970 Population']

# Calculate a column-wise differential. This means the data will
# now reflect change from the previous timepoint, rather than the
# total population at that time.
data = data.diff(axis=1)

# Put the base population back in
data['1970 Population'] = base_pop_1970

# Make a figure and axis that's big enough to show the data
fig, ax = plt.subplots(figsize=(12, 24))

# Plot a horizontal bar chart with pandas plotting method.
# Note the reversal of the data with [::-1], which is done
# to make sure the longest bars are at the top
data[::-1].plot(kind='barh', stacked=True, ax=ax)

# Add the numbers as text
for i, (country, row) in enumerate(data[::-1].iterrows()):
    pop = row.sum()
    text_pos = row[row>0].sum()
    ax.text(text_pos+1e7, i, '{:,}'.format(pop), va='center', fontsize=8)
    text_pos_neg = row[row < 0].abs().sum()
    if text_pos_neg > 0:
        ax.text(-(text_pos_neg+1e7), i, '{:,}'.format(text_pos_neg),␣
↪va='center', ha='right', fontsize=8)

# Add a vertical black line at zero as a visual aid
ax.axvline(0, 0, 1, c='k', lw=.5)

# Format the x axis
def billions_formatter(x, pos):
    return f'{x / 1000_000_000}'
ax.xaxis.set_major_formatter(plt.FuncFormatter(billions_formatter))

# Tweak axis
```

```
ax.set(
    ylabel='',
    title='Population change in the 117 most populated countries/territories␣
 ↪(1970-2022)',
    xlabel='Population in billions',
    xlim=(-.25e9, 1.75e9)
);

# Put legend in lower right of figure
ax.legend(loc='lower right')

# Save the figure with a tight bounding box and high resolution (300␣
 ↪dots-per-inch)
fig.savefig('../images/most_populated_countries_2022.png', bbox_inches='tight',␣
 ↪dpi=300)
```

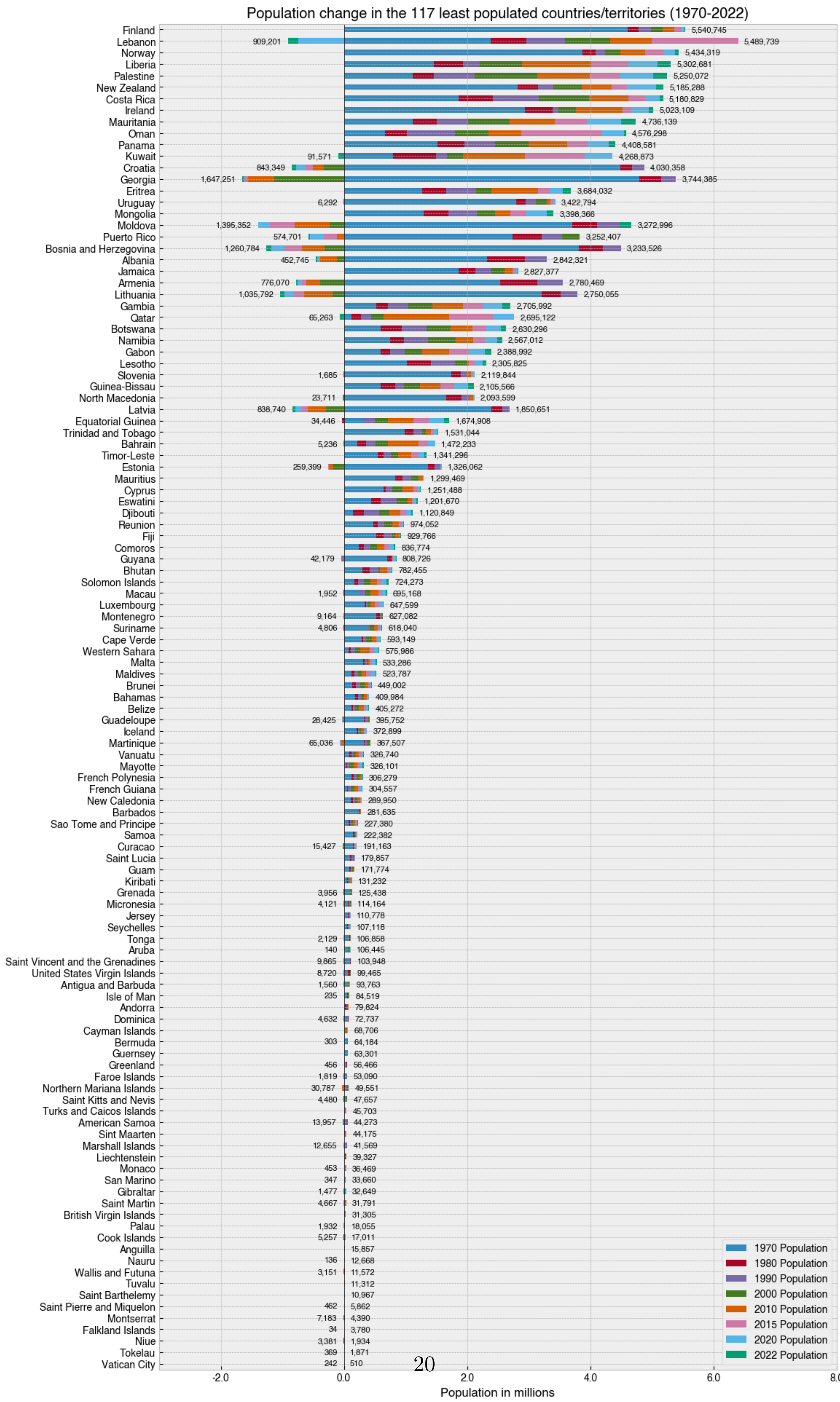


Population change in the 117 most populated countries/territories (1970-2022)
China 1,425,887,337
India 1,417,173,173
United States 338,289,857
Indonesia 275,501,339
Pakistan 235,824,862
Nigeria 218,541,212
Brazil 215,313,498
Bangladesh 171,186,372
Russia 5,667,120 144,713,314
Mexico 127,504,125
Japan 4,153,739 123,951,692
Ethiopia 123,379,924
Philippines 115,559,009
Egypt 110,990,103
DR Congo 99,010,212
Vietnam 98,186,856
Iran 88,550,570
Turkey 85,341,241
Germany 734,467 83,369,843
Thailand 71,697,030
United Kingdom 67,508,936
Tanzania 65,497,748
France 64,626,628
South Africa 59,893,885
Italy 1,195,432 59,037,474
Myanmar 54,179,306
Kenya 54,027,487
Colombia 51,874,024
South Korea 28,880 51,815,810
Spain 141,430 47,558,630
Uganda 47,249,585
Sudan 46,874,204
Argentina 45,510,318
Algeria 44,903,225
Iraq 44,496,122
Afghanistan 1,791,835 41,128,771
Poland 168,987 39,857,145
Ukraine 11,888,078 39,701,739
Canada 38,454,327
Morocco 37,457,971
Saudi Arabia 36,408,820
Angola 35,588,987
Uzbekistan 34,627,652
Peru 34,049,588
Malaysia 33,938,221
Yemen 33,696,614
Ghana 33,475,870
Mozambique 32,969,517
Nepal 30,547,580
Madagascar 29,611,714
Venezuela 2,228,020 28,301,696
Ivory Coast 28,160,542
Cameroon 27,914,536
Niger 26,207,977
Australia 26,177,413
North Korea 26,069,416
Taiwan 23,893,394
Burkina Faso 22,673,762
Mali 22,593,590
Syria 3,132,385 22,125,249
Sri Lanka 21,832,143
Malawi 20,405,317
Zambia 20,017,675
Romania 3,394,196 19,659,267
Chile 19,603,733
Kazakhstan 1,630,310 19,397,998
Ecuador 18,001,000
Guatemala 17,843,908
Chad 17,723,315
Somalia 17,597,511
Netherlands 17,564,014
Senegal 17,316,449
Cambodia 509,566 16,767,842
Zimbabwe 16,320,537
Guinea 13,859,341
Rwanda 13,776,698
Benin 13,352,864
Burundi 12,889,576
Tunisia 12,356,117
Bolivia 12,224,110
Belgium 11,655,930
Haiti 11,584,996
Jordan 11,285,869
Dominican Republic 11,228,821
Cuba 127,703 11,212,191
South Sudan 588,072 10,913,164
Sweden 10,549,347
Czech Republic 103,449 10,493,986
Honduras 10,432,860
Greece 653,138 10,384,971
Azerbaijan 10,358,074
Portugal 317,536 10,270,865
Papua New Guinea 10,142,619
Hungary 948,106 9,967,308
Tajikistan 9,952,787
Belarus 893,571 9,534,954
United Arab Emirates 9,441,129
Israel 9,038,309
Austria 8,939,617
Togo 8,848,699
Switzerland 8,740,472
Sierra Leone 8,605,718
Laos 7,529,475
Hong Kong 12,093 7,488,865
Serbia 766,164 7,221,365
Nicaragua 6,948,392
Libya 299,753 6,812,341
Bulgaria 2,198,653 6,781,953
Paraguay 6,780,744
Kyrgyzstan 6,630,623
Turkmenistan 6,430,770
El Salvador 6,336,392
Singapore 5,975,689
Republic of the Congo 5,970,424
Denmark 5,882,261
Slovakia 5,643,453
Central African Republic 5,579,144
-0.25 0.0 0.25 0.5 0.75 1.0 1.25 1.5 1.75
Population in billions
1970 Population
1980 Population
1990 Population
2000 Population
2010 Population
2015 Population
2020 Population
2022 Population

Rinse and repeat for the other half of the data. By using a second figure for the less-populated half of the dataset, we reset the scale on the x-axis and dramatically improve the explanatory power of the visualisations.

[34]:
```python
# Get the columns with the population data
pop_cols = df.columns[df.columns.str.endswith('Population')]

# Sort and select data
data = (df
 .set_index('Country')  # Set 'Country' as the index
 .sort_values('Rank', ascending=True)[pop_cols]  # Sort by 'Rank' and keep only␣
 ↪the columns with population data
 .iloc[-117:]  # Choose the 117 LEAST populated countries with location-based␣
 ↪indexing
 .loc[:, lambda df_: reversed(df_.columns)]  # Reverse the order of the columns␣
 ↪in a fancy way
)

# Pull out the base population at 1970
base_pop_1970 = data['1970 Population']

# Calculate a column-wise differential. This means the data will
# now reflect change from the previous timepoint, rather than the
# total population at that time.
data = data.diff(axis=1)

# Put the base population back in
data['1970 Population'] = base_pop_1970

# Make a figure and axis that's big enough to show the data
fig, ax = plt.subplots(figsize=(12, 24))

# Plot a horizontal bar chart with pandas plotting method.
# Note the reversal of the data with [::-1], which is done
# to make sure the longest bars are at the top
data[::-1].plot(kind='barh', stacked=True, ax=ax)

# Add the numbers as text
for i, (country, row) in enumerate(data[::-1].iterrows()):
    pop = row.sum()
    text_pos = row[row > 0].sum()
    ax.text(text_pos+1e5, i, '{:,}'.format(pop), va='center', fontsize=8)
    text_pos_neg = row[row < 0].abs().sum()
    if text_pos_neg > 0:
```

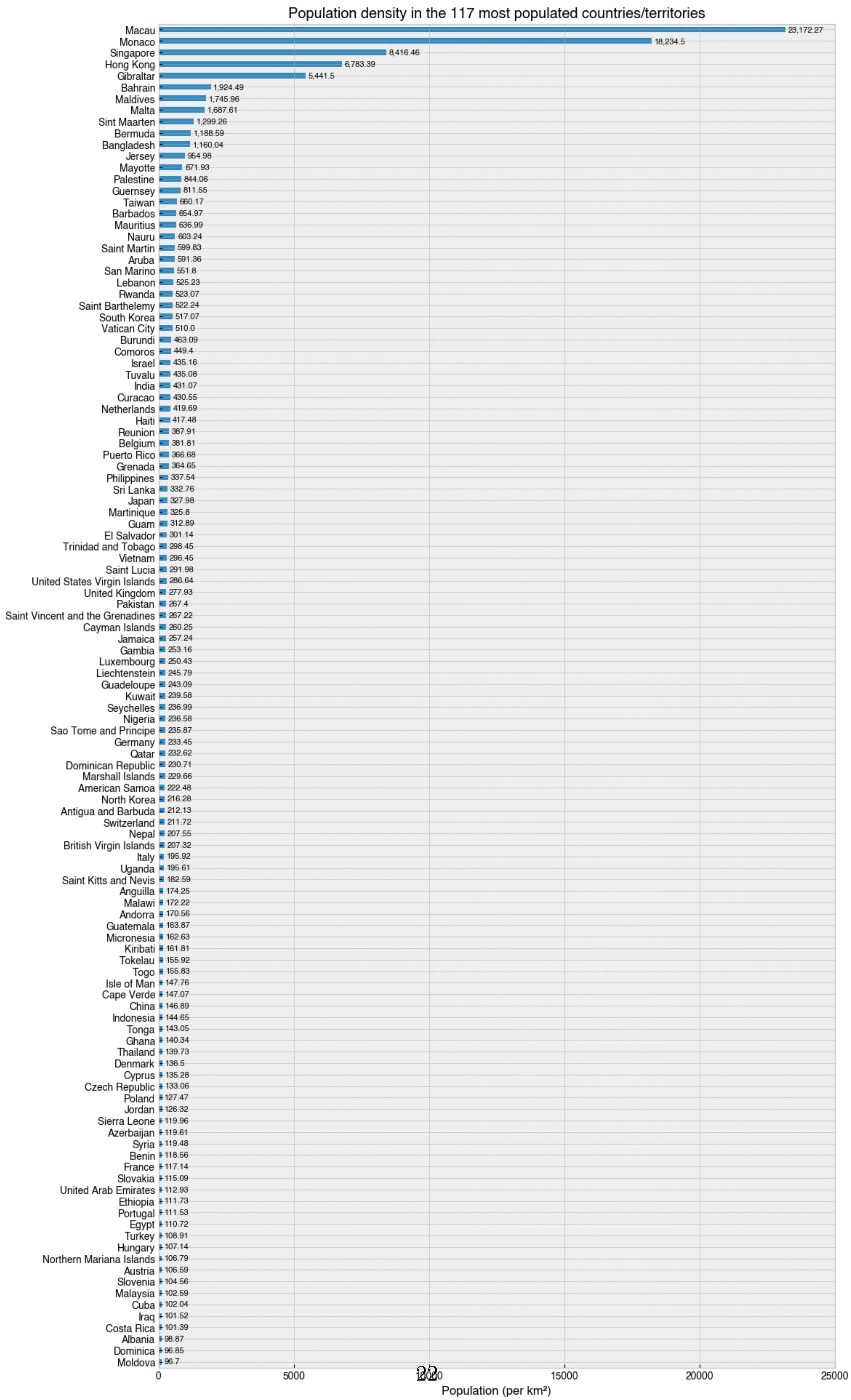
```
        ax.text(-(text_pos_neg+1e5), i, '{:,}'.format(text_pos_neg),␣
↪va='center', ha='right', fontsize=8)

# Add a vertical line at zero as a visual aid
ax.axvline(0, 0, 1, c='k', lw=.5)

# Format the x axis
def millions_formatter(x, pos):
    return f'{x / 1_000_000}'
ax.xaxis.set_major_formatter(plt.FuncFormatter(millions_formatter))

# Tweak axis
ax.set(
    ylabel='',
    title='Population change in the 117 least populated countries/territories␣
↪(1970-2022)',
    xlabel='Population in millions',
    xlim=(-3e6, 8e6)
)

# Put legend in lower right of figure
ax.legend(loc='lower right')

# Save the figure with a tight bounding box and high resolution (300␣
↪dots-per-inch)
fig.savefig('../images/least_populated_countries_2022.png',␣
↪bbox_inches='tight', dpi=300)
```


Population change in the 117 least populated countries/territories (1970-2022)
Finland 5,540,745
Lebanon 909,201 5,489,739
Norway 5,434,319
Liberia 5,302,681
Palestine 5,250,072
New Zealand 5,185,288
Costa Rica 5,180,829
Ireland 5,023,109
Mauritania 4,736,139
Oman 4,576,298
Panama 4,408,581
Kuwait 91,571 4,268,873
Croatia 843,349 4,030,358
Georgia 1,647,251 3,744,385
Eritrea 3,684,032
Uruguay 6,292 3,422,794
Mongolia 3,398,366
Moldova 1,395,352 3,272,996
Puerto Rico 574,701 3,252,407
Bosnia and Herzegovina 1,260,784 3,233,526
Albania 452,745 2,842,321
Jamaica 2,827,377
Armenia 776,070 2,780,469
Lithuania 1,035,792 2,750,055
Gambia 2,705,992
Qatar 65,263 2,695,122
Botswana 2,630,296
Namibia 2,567,012
Gabon 2,388,992
Lesotho 2,305,825
Slovenia 1,685 2,119,844
Guinea-Bissau 2,105,566
North Macedonia 23,711 2,093,599
Latvia 838,740 1,850,651
Equatorial Guinea 34,446 1,674,908
Trinidad and Tobago 1,531,044
Bahrain 5,236 1,472,233
Timor-Leste 1,341,296
Estonia 259,399 1,326,062
Mauritius 1,299,469
Cyprus 1,251,488
Eswatini 1,201,670
Djibouti 1,120,849
Reunion 974,052
Fiji 929,766
Comoros 836,774
Guyana 42,179 808,726
Bhutan 782,455
Solomon Islands 724,273
Macau 1,952 695,168
Luxembourg 647,599
Montenegro 9,164 627,082
Suriname 4,806 618,040
Cape Verde 593,149
Western Sahara 575,986
Malta 533,286
Maldives 523,787
Brunei 449,002
Bahamas 409,984
Belize 405,272
Guadeloupe 28,425 395,752
Iceland 372,899
Martinique 65,036 367,507
Vanuatu 326,740
Mayotte 326,101
French Polynesia 306,279
French Guiana 304,557
New Caledonia 289,950
Barbados 281,635
Sao Tome and Principe 227,380
Samoa 222,382
Curacao 15,427 191,163
Saint Lucia 179,857
Guam 171,774
Kiribati 131,232
Grenada 3,956 125,438
Micronesia 4,121 114,164
Jersey 110,778
Seychelles 107,118
Tonga 2,129 106,858
Aruba 140 106,445
Saint Vincent and the Grenadines 9,865 103,948
United States Virgin Islands 8,720 99,465
Antigua and Barbuda 1,560 93,763
Isle of Man 235 84,519
Andorra 79,824
Dominica 4,632 72,737
Cayman Islands 68,706
Bermuda 303 64,184
Guernsey 63,301
Greenland 456 56,466
Faroe Islands 1,819 53,090
Northern Mariana Islands 30,787 49,551
Saint Kitts and Nevis 4,480 47,657
Turks and Caicos Islands 45,703
American Samoa 13,957 44,273
Sint Maarten 44,175
Marshall Islands 12,655 41,569
Liechtenstein 39,327
Monaco 453 36,469
San Marino 347 33,660
Gibraltar 1,477 32,649
Saint Martin 4,667 31,791
British Virgin Islands 31,305
Palau 1,932 18,055
Cook Islands 5,257 17,011
Anguilla 15,857
Nauru 136 12,668
Wallis and Futuna 3,151 11,572
Tuvalu 11,312
Saint Barthelemy 10,967
Saint Pierre and Miquelon 462 5,862
Montserrat 7,183 4,390
Falkland Islands 34 3,780
Niue 3,381 1,934
Tokelau 369 1,871
Vatican City 242 510
-2.0
0.0
2.0
4.0
6.0
8.0
Population in millions
1970 Population
1980 Population
1990 Population
2000 Population
2010 Population
2015 Population
2020 Population
2022 Population

1.7 The big picture - population density

After plotting the above, I realised I could do the same using only the `Density (per km²)` column. This was a little bit easier as it didn't require stacking the bars.

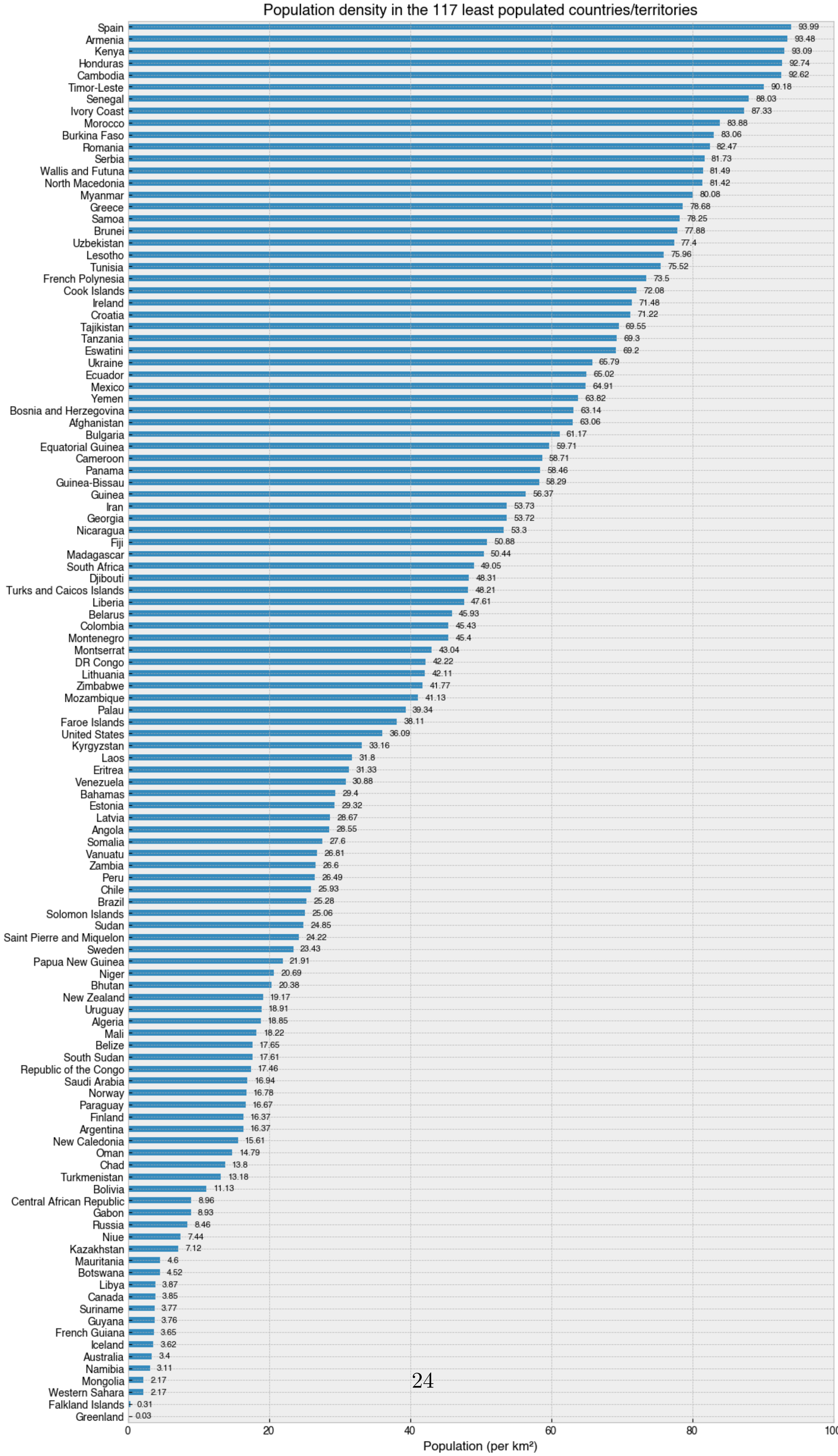
[35]:
```
# Sort and select data
data = (df
 .set_index('Country')  # Set 'Country' as the index
 .sort_values('Density (per km²)', ascending=True)
 .iloc[-117:]  # Choose the 117 MOST densely populated countries with␣
 ↪location-based indexing
)

# Make a figure and axis that's big enough to show the data
fig, ax = plt.subplots(figsize=(12, 24))

# Plot a horizontal bar chart with pandas plotting method.
data.plot(kind='barh', y='Density (per km²)', ax=ax, legend=False)

# Add the numbers in text
for i, (country, row) in enumerate(data.iterrows()):
    pop = row['Density (per km²)']
    ax.text(pop+100, i, '{:,}'.format(round(pop, 2)), va='center', fontsize=8)

# Add a vertical line at zero as a visual aid
ax.axvline(0, 0, 1, c='k', lw=.5)

# Tweak axis
ax.set(
    ylabel='',
    title='Population density in the 117 most populated countries/territories',
    xlabel='Population (per km²)',
    xlim=(0, 25000)
)

# Save the figure with a tight bounding box and high resolution (300␣
 ↪dots-per-inch)
fig.savefig('../images/most_dense_countries_2022.png', bbox_inches='tight',␣
 ↪dpi=300)
```


Population density in the 117 most populated countries/territories
Macau 23,172.27
Monaco 18,234.5
Singapore 8,416.46
Hong Kong 6,783.39
Gibraltar 5,441.5
Bahrain 1,924.49
Maldives 1,745.96
Malta 1,687.61
Sint Maarten 1,299.26
Bermuda 1,188.59
Bangladesh 1,160.04
Jersey 954.98
Mayotte 871.93
Palestine 844.06
Guernsey 811.55
Taiwan 660.17
Barbados 654.97
Mauritius 636.99
Nauru 603.24
Saint Martin 599.83
Aruba 591.36
San Marino 551.8
Lebanon 525.23
Rwanda 523.07
Saint Barthelemy 522.24
South Korea 517.07
Vatican City 510.0
Burundi 463.09
Comoros 449.4
Israel 435.16
Tuvalu 435.08
India 431.07
Curacao 430.55
Netherlands 419.69
Haiti 417.48
Reunion 387.91
Belgium 381.81
Puerto Rico 366.68
Grenada 364.65
Philippines 337.54
Sri Lanka 332.76
Japan 327.98
Martinique 325.8
Guam 312.89
El Salvador 301.14
Trinidad and Tobago 298.45
Vietnam 296.45
Saint Lucia 291.98
United States Virgin Islands 286.64
United Kingdom 277.93
Pakistan 267.4
Saint Vincent and the Grenadines 267.22
Cayman Islands 260.25
Jamaica 257.24
Gambia 253.16
Luxembourg 250.43
Liechtenstein 245.79
Guadeloupe 243.09
Kuwait 239.58
Seychelles 236.99
Nigeria 236.58
Sao Tome and Principe 235.87
Germany 233.45
Qatar 232.62
Dominican Republic 230.71
Marshall Islands 229.66
American Samoa 222.48
North Korea 216.28
Antigua and Barbuda 212.13
Switzerland 211.72
Nepal 207.55
British Virgin Islands 207.32
Italy 195.92
Uganda 195.61
Saint Kitts and Nevis 182.59
Anguilla 174.25
Malawi 172.22
Andorra 170.56
Guatemala 163.87
Micronesia 162.63
Kiribati 161.81
Tokelau 155.92
Togo 155.83
Isle of Man 147.76
Cape Verde 147.07
China 146.89
Indonesia 144.65
Tonga 143.05
Ghana 140.34
Thailand 139.73
Denmark 136.5
Cyprus 135.28
Czech Republic 133.06
Poland 127.47
Jordan 126.32
Sierra Leone 119.96
Azerbaijan 119.61
Syria 119.48
Benin 118.56
France 117.14
Slovakia 115.09
United Arab Emirates 112.93
Ethiopia 111.73
Portugal 111.53
Egypt 110.72
Turkey 108.91
Hungary 107.14
Northern Mariana Islands 106.79
Austria 106.59
Slovenia 104.56
Malaysia 102.59
Cuba 102.04
Iraq 101.52
Costa Rica 101.39
Albania 98.87
Dominica 96.85
Moldova 96.7
0
5000
10000
15000
20000
25000
Population (per km²)

What's it like to be one of 23,172 people in a square kilometer of land? Go to Macau and you'll find out. Also, I didn't realise Gibralter was so dense.

Now for the other half...

[36]:
```python
# Sort and select data
data = (df
 .set_index('Country')  # Set 'Country' as the index
 .sort_values('Density (per km²)', ascending=True)
 .iloc[0:117]  # Choose the 117 LEAST densely populated countries with location-based indexing
)

# Make a figure and axis that's big enough to show the data
fig, ax = plt.subplots(figsize=(12, 24))

# Plot a horizontal bar chart with pandas plotting method.
data.plot(kind='barh', y='Density (per km²)', ax=ax, legend=False)

# Add the numbers in text
for i, (country, row) in enumerate(data.iterrows()):
    pop = row['Density (per km²)']
    ax.text(pop+1, i, '{:,}'.format(round(pop, 2)), va='center', fontsize=8)

# Add a vertical line at zero as a visual aid
ax.axvline(0, 0, 1, c='k', lw=.5)

# Tweak axis
ax.set(
    ylabel='',
    title='Population density in the 117 least populated countries/territories',
    xlabel='Population (per km²)',
    xlim=(0, 100)
)

# Save the figure with a tight bounding box and high resolution (300 dots-per-inch)
fig.savefig('../images/least_dense_countries_2022.png', bbox_inches='tight', dpi=300)
```


Population density in the 117 least populated countries/territories
Spain 93.99
Armenia 93.48
Kenya 93.09
Honduras 92.74
Cambodia 92.62
Timor-Leste 90.18
Senegal 88.03
Ivory Coast 87.33
Morocco 83.88
Burkina Faso 83.06
Romania 82.47
Serbia 81.73
Wallis and Futuna 81.49
North Macedonia 81.42
Myanmar 80.08
Greece 78.68
Samoa 78.25
Brunei 77.88
Uzbekistan 77.4
Lesotho 75.96
Tunisia 75.52
French Polynesia 73.5
Cook Islands 72.08
Ireland 71.48
Croatia 71.22
Tajikistan 69.55
Tanzania 69.3
Eswatini 69.2
Ukraine 65.79
Ecuador 65.02
Mexico 64.91
Yemen 63.82
Bosnia and Herzegovina 63.14
Afghanistan 63.06
Bulgaria 61.17
Equatorial Guinea 59.71
Cameroon 58.71
Panama 58.46
Guinea-Bissau 58.29
Guinea 56.37
Iran 53.73
Georgia 53.72
Nicaragua 53.3
Fiji 50.88
Madagascar 50.44
South Africa 49.05
Djibouti 48.31
Turks and Caicos Islands 48.21
Liberia 47.61
Belarus 45.93
Colombia 45.43
Montenegro 45.4
Montserrat 43.04
DR Congo 42.22
Lithuania 42.11
Zimbabwe 41.77
Mozambique 41.13
Palau 39.34
Faroe Islands 38.11
United States 36.09
Kyrgyzstan 33.16
Laos 31.8
Eritrea 31.33
Venezuela 30.88
Bahamas 29.4
Estonia 29.32
Latvia 28.67
Angola 28.55
Somalia 27.6
Vanuatu 26.81
Zambia 26.6
Peru 26.49
Chile 25.93
Brazil 25.28
Solomon Islands 25.06
Sudan 24.85
Saint Pierre and Miquelon 24.22
Sweden 23.43
Papua New Guinea 21.91
Niger 20.69
Bhutan 20.38
New Zealand 19.17
Uruguay 18.91
Algeria 18.85
Mali 18.22
Belize 17.65
South Sudan 17.61
Republic of the Congo 17.46
Saudi Arabia 16.94
Norway 16.78
Paraguay 16.67
Finland 16.37
Argentina 16.37
New Caledonia 15.61
Oman 14.79
Chad 13.8
Turkmenistan 13.18
Bolivia 11.13
Central African Republic 8.96
Gabon 8.93
Russia 8.46
Niue 7.44
Kazakhstan 7.12
Mauritania 4.6
Botswana 4.52
Libya 3.87
Canada 3.85
Suriname 3.77
Guyana 3.76
French Guiana 3.65
Iceland 3.62
Australia 3.4
Namibia 3.11
Mongolia 2.17
Western Sahara 2.17
Falkland Islands 0.31
Greenland 0.03
0
20
40
60
80
100
Population (per km²)

Iceland… A beautiful country, where a kilometer of land is shared by only 3.62 people.

That's it for now. These examples are complicated, but if you study them carefully, change bits, and come up with your own variations, you will learn a lot about Python and `pandas` in the process! If you are feeling adventures, why not try and replicate these plots for the `Area (km²)` column? In this situation, countries like Russia and Canada would be right at the top, and at the bottom would be The Vatican City and small island nations like the Falkland and Faroe Islands.