ids-pdl09-hwk.ipynb: This Jupyter notebook is provided by Joachim Vogt for the *Python Data Lab* of the module *Introduction to Data Science* offered in Fall 2022 at Jacobs University Bremen. Module instructors are Hilke Brockmann, Adalbert Wilhelm, and Joachim Vogt. Jupyter notebooks and other learning resources are available from a dedicated *module platform*.

Homework assignments: Working with Pandas

The homework assignments in this notebook supplement the tutorial *Working with Pandas*.

- Solve the assignments according to the instructions.
- Upload the completed notebook to the module platform.
- Do not forget to enter your name in the markdown cell below.

The homework set carries a total of 20 points. Square brackets in the assignment titles specify individual point contributions.

Name: Hamza Rehman

Preparation

Import NumPy, `pyplot` from matplotlib, and Pandas as usual.

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
```

The following data files are expected to reside in the working directory. Identify the files on the module platform and upload them to the same folder as this Jupyter notebook.

- `gdp-per-capita-in-us-dollar-world-bank.csv` : GDP per capita in constant 2010 US dollars 1960-2020, published by the World Bank, 2021-07-30, available from Our World in Data.
- `life-expectancy-at-birth-total-years.csv` : Life expectancy at birth 1960-2019, published by the World Bank, 2021-07-30, available from Our World in Data.
- `leb_gdpint_decade.png` : graphics of a table to be reproduced in the assignment *Pivot tables*.

Assignment: Hierarchical indexing [5]

According to Wikipedia (accessed on 2022-07-26), the resident numbers of Berlin, Cologne, Hamburg, Munich in the years 1980, 2000, 2020/21 were as follows.

City	1980	2000	2020/21
Berlin	3048759	3382169	3677472
Cologne	976694	962884	1073096
Hamburg	1645095	1715392	1853935
Munich	1298941	1210223	1487708

In the cell below,

1. store the three sets of resident numbers for the years 1980, 2000, 2020/21 in a single Pandas DataFrame object `ResA` indexed by the city names,
2. apply the function `stack()` to `ResA` and store the result in a Pandas Series object `ResB`,
3. from `ResB` obtain the index (a MultiIndex object) and store it as `IndB`,
4. using the Pandas function `MultiIndex.from_arrays()`, construct a MultiIndex object `IndC` identical to `IndB`,
5. using the Pandas function `MultiIndex.from_product()`, construct a MultiIndex object `IndD` identical to `IndB`.

In [14]:
```
### Construct and display Pandas DataFrame ResA.
ResA = pd.DataFrame( {'1980':[3048759,976694,1645095,1298941],
                      '2000':[3382169,962884,1715392,1210223],
                      '2020/21':[3677472,1073096,1853935,1487708]},
                    index=['Berlin','Cologne','Hamburg','Munich'])
display(ResA)

### Stack ResA to obtain ResB.
ResB = ResA.stack()
display(ResB)
```

```python
### Store the index of ResB in variable IndB, then display.
IndB = ResB.index
display(IndB)

### Construct and display MultiIndex object IndC from arrays.
cities = ['Berlin','Berlin','Berlin','Cologne','Cologne','Cologne','Hamburg','Hamburg','Hamburg','Munich',
          'Munich','Munich']
years = ['1980','2000','2020/21','1980','2000','2020/21','1980','2000','2020/21','1980','2000','2020/21']
IndC = pd.MultiIndex.from_arrays([cities,years])
display(IndC)

### Construct and display MultiIndex object IndD from product.
IndD = pd.MultiIndex.from_product([['Berlin','Cologne','Hamburg','Munich'], ['1980','2000','2020/21']])
display(IndD)
```

	1980	2000	2020/21
Berlin	3048759	3382169	3677472
Cologne	976694	962884	1073096
Hamburg	1645095	1715392	1853935
Munich	1298941	1210223	1487708

```
Berlin   1980       3048759
         2000       3382169
         2020/21    3677472
Cologne  1980        976694
         2000        962884
         2020/21    1073096
Hamburg  1980       1645095
         2000       1715392
         2020/21    1853935
Munich   1980       1298941
         2000       1210223
         2020/21    1487708
dtype: int64
MultiIndex([( 'Berlin',    '1980'),
            ( 'Berlin',    '2000'),
            ( 'Berlin', '2020/21'),
            ('Cologne',    '1980'),
            ('Cologne',    '2000'),
            ('Cologne', '2020/21'),
            ('Hamburg',    '1980'),
            ('Hamburg',    '2000'),
            ('Hamburg', '2020/21'),
            ( 'Munich',    '1980'),
            ( 'Munich',    '2000'),
            ( 'Munich', '2020/21')],
           )
MultiIndex([( 'Berlin',    '1980'),
            ( 'Berlin',    '2000'),
            ( 'Berlin', '2020/21'),
            ('Cologne',    '1980'),
            ('Cologne',    '2000'),
            ('Cologne', '2020/21'),
            ('Hamburg',    '1980'),
            ('Hamburg',    '2000'),
            ('Hamburg', '2020/21'),
            ( 'Munich',    '1980'),
            ( 'Munich',    '2000'),
            ( 'Munich', '2020/21')],
           )
MultiIndex([( 'Berlin',    '1980'),
            ( 'Berlin',    '2000'),
            ( 'Berlin', '2020/21'),
            ('Cologne',    '1980'),
            ('Cologne',    '2000'),
            ('Cologne', '2020/21'),
            ('Hamburg',    '1980'),
            ('Hamburg',    '2000'),
            ('Hamburg', '2020/21'),
            ( 'Munich',    '1980'),
            ( 'Munich',    '2000'),
            ( 'Munich', '2020/21')],
           )
```

Assignment: GroupBy mechanism [8]

The file `gdp-per-capita-in-us-dollar-world-bank.csv` contains data on GDP per capita in constant 2010 US dollars 1960-2020, as published by the World Bank on 2021-07-30, and made available through Our World in Data.

- Click on the filename in the directory listing to display the content of this comma-separated text file to study the structure.
- Consult the associated tutorial notebook `ids-pdl09-tut.ipynb` and study how the data from the file `life-expectancy-at-birth-total-years.csv` are processed.

- The same processing steps are to be applied to the GDP per capita data from the file `gdp-per-capita-in-us-dollar-world-bank.csv`. Details are given below.

Using the Pandas function `read_csv()`, the data are loaded and stored in a DataFrame.

In [15]:
```
gdp_full = pd.read_csv('gdp-per-capita-in-us-dollar-world-bank.csv')
display(gdp_full)
```

	Entity	Code	Year	GDP per capita (constant 2010 US$)
0	Afghanistan	AFG	2002	330.303494
1	Afghanistan	AFG	2003	343.080890
2	Afghanistan	AFG	2004	333.216617
3	Afghanistan	AFG	2005	357.234762
4	Afghanistan	AFG	2006	365.284371
...	...	...	...	...
12147	Zimbabwe	ZWE	2016	1224.314460
12148	Zimbabwe	ZWE	2017	1263.278346
12149	Zimbabwe	ZWE	2018	1289.146499
12150	Zimbabwe	ZWE	2019	1168.008072
12151	Zimbabwe	ZWE	2020	1058.845827

12152 rows × 4 columns

From the full DataFrame `gdp_full`, remove rows for entities that are not single countries but world regions.

In [16]:
```
gdp_full.dropna(inplace=True)
gdp_full = gdp_full[gdp_full["Entity"]!="World"]
display(gdp_full)
```

	Entity	Code	Year	GDP per capita (constant 2010 US$)
0	Afghanistan	AFG	2002	330.303494
1	Afghanistan	AFG	2003	343.080890
2	Afghanistan	AFG	2004	333.216617
3	Afghanistan	AFG	2005	357.234762
4	Afghanistan	AFG	2006	365.284371
...	...	...	...	...
12147	Zimbabwe	ZWE	2016	1224.314460
12148	Zimbabwe	ZWE	2017	1263.278346
12149	Zimbabwe	ZWE	2018	1289.146499
12150	Zimbabwe	ZWE	2019	1168.008072
12151	Zimbabwe	ZWE	2020	1058.845827

9525 rows × 4 columns

Rename the GDP per capita column label to `'GDP/cap.'`.

In [17]:
```
gdp_full.rename(columns = {gdp_full.columns[3] : 'GDP/cap.'},inplace = True, errors= 'raise')
display(gdp_full.head)
```

```
<bound method NDFrame.head of             Entity Code  Year     GDP/cap.
0      Afghanistan  AFG  2002   330.303494
1      Afghanistan  AFG  2003   343.080890
2      Afghanistan  AFG  2004   333.216617
3      Afghanistan  AFG  2005   357.234762
4      Afghanistan  AFG  2006   365.284371
...            ...  ...   ...          ...
12147     Zimbabwe  ZWE  2016  1224.314460
12148     Zimbabwe  ZWE  2017  1263.278346
12149     Zimbabwe  ZWE  2018  1289.146499
12150     Zimbabwe  ZWE  2019  1168.008072
12151     Zimbabwe  ZWE  2020  1058.845827

[9525 rows x 4 columns]>
```

Store the column of GDP per capita values.

In [18]:
```
gdp_per_capita_values = gdp_full['GDP/cap.'].values
```

Construct the MultiIndex object from the `'Code'` and `'Year'` columns.

In [19]:
```
codes = gdp_full['Code'].values
years = gdp_full['Year'].values
mulind = pd.MultiIndex.from_arrays([codes,years])
print(mulind)
```

```
MultiIndex([('AFG', 2002),
            ('AFG', 2003),
            ('AFG', 2004),
            ('AFG', 2005),
            ('AFG', 2006),
            ('AFG', 2007),
            ('AFG', 2008),
            ('AFG', 2009),
            ('AFG', 2010),
            ('AFG', 2011),
            ...
            ('ZWE', 2011),
            ('ZWE', 2012),
            ('ZWE', 2013),
            ('ZWE', 2014),
            ('ZWE', 2015),
            ('ZWE', 2016),
            ('ZWE', 2017),
            ('ZWE', 2018),
            ('ZWE', 2019),
            ('ZWE', 2020)],
           length=9525)
```

Using the array of GDP per capita values with the MultiIndex object as an index, we construct a Series object that is then unstacked to yield a DataFrame in the desired format.

In [20]:
```
nDF = pd.Series(gdp_per_capita_values,index = mulind).unstack()
display(nDF)
```

	1960	1961	1962	1963	1964	1965	1966	1967	1968	1969	...
ABW	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
AFG	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
AGO	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
ALB	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
AND	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
...	...	...	...	...	...	...	...	...	...	...	...
WSM	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
YEM	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
ZAF	4624.077033	4685.490949	4852.780839	5081.233260	5347.340956	5531.731402	5631.010299	5882.849706	5970.578833	6091.031057	...
ZMB	1495.703272	1471.244667	1392.222249	1395.225016	1519.023893	1718.675662	1573.798595	1646.596597	1615.678326	1558.175421	...
ZWE	994.698381	1022.764137	1002.974747	1030.026148	984.670135	998.754210	980.595168	1027.852205	1013.728418	1101.938805	...

206 rows × 61 columns

Rows and columns are swapped through the application of `transpose()`.

In [21]:
```
nDF = nDF.transpose()
display(nDF.head())
```

	ABW	AFG	AGO	ALB	AND	ARE	ARG	ARM	ASM	ATG	...	VCT	VEN	VIR	VNM	VUT	WSM	YEM	
1960	NaN	NaN	NaN	NaN	NaN	NaN	5642.764587	NaN	NaN	NaN	...	1752.014362	12457.210022	NaN	NaN	NaN	NaN	NaN	462
1961	NaN	NaN	NaN	NaN	NaN	NaN	5853.170781	NaN	NaN	NaN	...	1804.632391	12401.867700	NaN	NaN	NaN	NaN	NaN	468
1962	NaN	NaN	NaN	NaN	NaN	NaN	5711.182038	NaN	NaN	NaN	...	1847.223030	12992.822489	NaN	NaN	NaN	NaN	NaN	485
1963	NaN	NaN	NaN	NaN	NaN	NaN	5323.493318	NaN	NaN	NaN	...	1711.726256	13037.601796	NaN	NaN	NaN	NaN	NaN	508
1964	NaN	NaN	NaN	NaN	NaN	NaN	5772.649438	NaN	NaN	NaN	...	1755.424051	13999.206299	NaN	NaN	NaN	NaN	NaN	534

5 rows × 206 columns

Apply the `describe()` method to obtain summary statistics of a DataFrame.

In [22]:
```
display(nDF.describe().transpose().head())
```

	count	mean	std	min	25%	50%	75%	max
ABW	32.0	25413.523448	2612.067368	15669.365906	24616.439346	26280.841687	26986.424676	28413.318599
AFG	19.0	487.286078	101.190910	330.303494	385.416630	543.302967	572.366373	587.565035
AGO	41.0	2887.765429	585.847171	1733.843074	2433.861819	2902.616443	3409.927901	3843.198733
ALB	41.0	2873.909253	1234.499851	1243.605824	1986.321914	2244.624632	4094.350334	5207.305322
AND	51.0	40090.960490	4264.182125	32296.694560	35932.887541	40850.248143	42669.429552	47844.218979

Apply the GroupBy mechanism to aggregate the GDP per capita data from the DataFrame `gdp_full` in decades, using `min`, `median`, and `max` as aggregation functions.

Hint: Consult section 3.08 Aggregation and Grouping of the Python Data Science Handbook by Jake Vanderplas for an effective implementation of decade aggregation.

In [23]:

```
decade = 10 * (gdp_full['Year'] // 10)
decade = decade.astype(str) + 's'
decade.name = 'Decade'
gdp_full.groupby(decade)['GDP/cap.'].aggregate(['min', 'mean', 'max'])
```

Out[23]:

	min	mean	max
Decade			
1960s	132.077606	5446.641045	48285.921744
1970s	168.190361	9375.531913	109318.189687
1980s	164.464529	10276.686062	121971.980592
1990s	164.336583	11100.847543	134910.889174
2000s	194.873078	14264.360368	170539.637934
2010s	208.074775	16058.684153	209224.505501
2020s	202.372052	13406.552083	107457.984960

Assignment: Pivot tables [7]

The pivot table below shows the median life expectancy at birth in the six decades since 1960 for selected intervals of GDP per capita in 2010 US$. As before, all statistics are based on the country distributions given in the files `life-expectancy-at-birth-total-years.csv` and `gdp-per-capita-in-us-dollar-world-bank.csv`.

Follow the instructions below to reproduce this pivot table.

Using the Pandas functions `read_csv()` and `merge()`, join the contents of the files `life-expectancy-at-birth-total-years.csv` and `gdp-per-capita-in-us-dollar-world-bank.csv` to obtain a single Pandas DataFrame object `leb_gdp`. Drop undefined data and world regions, keeping only data from indvidual countries, and rename inconveniently long column labels. See the first session of the Python Data Lab for an implementation of this sequence of operations.

In [28]:

```
leb_full = pd.read_csv('life-expectancy-at-birth-total-years.csv')
gdp_full = pd.read_csv('gdp-per-capita-in-us-dollar-world-bank.csv')
leb_gdp = leb_full.merge(gdp_full)

leb_gdp.dropna(inplace = True)
leb_gdp = leb_gdp[leb_gdp['Entity'] != 'World']

leb_gdp.rename(columns = {leb_gdp.columns[3] : 'Life exp.', leb_gdp.columns[4] : 'GDP/cap'},inplace = True, err
display(leb_gdp)
```

	Entity	Code	Year	Life exp.	GDP/cap
0	Afghanistan	AFG	2002	56.784	330.303494
1	Afghanistan	AFG	2003	57.271	343.080890
2	Afghanistan	AFG	2004	57.772	333.216617
3	Afghanistan	AFG	2005	58.290	357.234762
4	Afghanistan	AFG	2006	58.826	365.284371
...	...	...	...	...	...
11474	Zimbabwe	ZWE	2015	59.534	1234.102191
11475	Zimbabwe	ZWE	2016	60.294	1224.314460
11476	Zimbabwe	ZWE	2017	60.812	1263.278346
11477	Zimbabwe	ZWE	2018	61.195	1289.146499
11478	Zimbabwe	ZWE	2019	61.490	1168.008072

8899 rows × 5 columns

Construct the variable `decade` from the `Year` column of the DataFrame `leb_gdp`.

In [26]:
```
decade = 10 * (leb_gdp['Year'] // 10)
decade = decade.astype(str) + 's'
decade.name = 'Decade'
```

Using the Pandas function `cut()`, define partitions of the GDP per capita data column in `leb_gdp` into intervals with boundaries `0,300,1000,3000,10000,30000,100000,300000`, and save them in the variable `gdp_partition`.

In [27]:
```
gdp_partition = pd.cut(leb_gdp['GDP/cap'], [0, 300, 1000, 3000, 10000, 30000, 100000, 300000])
display(gdp_partition)
```

```
0         (300, 1000]
1         (300, 1000]
2         (300, 1000]
3         (300, 1000]
4         (300, 1000]
             ...
11474    (1000, 3000]
11475    (1000, 3000]
11476    (1000, 3000]
11477    (1000, 3000]
11478    (1000, 3000]
Name: GDP/cap, Length: 8899, dtype: category
Categories (7, interval[int64, right]): [(0, 300] < (300, 1000] < (1000, 3000] < (3000, 10000] < (10000, 30000]
< (30000, 100000] < (100000, 300000]]
```

Call the Pandas function `pivot_table()` on the life expectancy at birth column as key variable, with `decade` and `gdp_partition` as further arguments, and `median` as the aggregation function.

In [29]:
```
l=leb_gdp.pivot_table(leb_gdp ,decade, gdp_partition, aggfunc = 'median')
display(l)
```

							GDP/cap					
GDP/cap	**(0, 300]**	**(300, 1000]**	**(1000, 3000]**	**(3000, 10000]**	**(10000, 30000]**	**(30000, 100000]**	**(100000, 300000]**	**(0, 300]**	**(300, 1000]**	**(1000, 3000]**	**...**	**(10 30**
Decade												
1960s	237.236614	619.652614	1510.540607	4898.340218	17102.594040	30736.683999	NaN	42.4605	46.1025	50.4795	...	70.62
1970s	263.458488	609.165819	1779.605218	5228.366062	20559.723422	37973.160753	103686.021950	46.3270	46.8340	55.9785	...	71.97
1980s	223.634924	548.404407	1734.989400	4440.105730	18549.701032	37741.078394	114664.571682	46.4305	49.9220	60.7920	...	73.90
1990s	219.533511	543.142279	1665.354633	4898.700341	20065.078348	37354.951853	NaN	48.1940	53.6330	65.2510	...	74.51
2000s	229.699443	600.455777	1755.418682	5209.038698	17616.650826	42859.796489	104943.439300	52.7560	56.1780	66.5390	...	75.21
2010s	231.333990	662.940658	1652.661261	5394.996711	15930.661772	47275.403025	106648.366503	59.8940	60.9935	66.8955	...	76.48

6 rows × 21 columns