

PANDAS 001 - Transforming DataFrames

Inspecting a DataFrame

When you get a new DataFrame to work with, the first thing you need to do is explore it and see what it contains. There are several useful methods and attributes for this.

`.head()` returns the first few rows (the “head” of the DataFrame).

`.info()` shows information on each of the columns, such as the data type and number of missing values.

`.shape` returns the number of rows and columns of the DataFrame.

`.describe()` calculates a few summary statistics for each column.

`homelessness` is a DataFrame containing estimates of homelessness in each U.S. state in 2018. The `individuals` column is the number of homeless individuals not part of a family with children. The `family_members` column is the number of homeless individuals part of a family with children. The `state_pop` column is the state's total population.

Import Pandas

Print the head of the `homelessness` DataFrame.

In [21]:

```
import pandas as pd

#load data
homelessness = pd.read_csv('homelessness.csv') #Path to the dataset(file)

# Print the head of the homelessness data
print(homelessness.head())
```

	Unnamed: 0		region	state	individuals	family_members	\
0	0	East South Central	Alabama	2570.0	864.0		
1	1	Pacific	Alaska	1434.0	582.0		
2	2	Mountain	Arizona	7259.0	2606.0		
3	3	West South Central	Arkansas	2280.0	432.0		
4	4	Pacific	California	109008.0	20964.0		

	state_pop
0	4887681
1	735139
2	7158024
3	3009733
4	39461588

Print information about the column types and missing values in `homelessness`

In [22]:

```
# Print information about homelessness
print(homelessness.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  -
 0   Unnamed: 0      51 non-null    int64  
 1   region          51 non-null    object  
 2   state           51 non-null    object  
 3   individuals     51 non-null    float64 
 4   family_members  51 non-null    float64 
 5   state_pop       51 non-null    int64  
dtypes: float64(2), int64(2), object(2)
memory usage: 2.5+ KB
None
```

Print the number of rows and columns in homelessness

In [23]:

```
# Print the shape of homelessness
print(homelessness.shape)

(51, 6)
```

Print some summary statistics that describe the homelessness DataFrame.

In [24]:

```
#summary statistics
print(homelessness.describe())
```

	Unnamed: 0	individuals	family_members	state_pop
count	51.000000	51.000000	51.000000	5.100000e+01
mean	25.000000	7225.784314	3504.882353	6.405637e+06
std	14.866069	15991.025083	7805.411811	7.327258e+06
min	0.000000	434.000000	75.000000	5.776010e+05
25%	12.500000	1446.500000	592.000000	1.777414e+06
50%	25.000000	3082.000000	1482.000000	4.461153e+06
75%	37.500000	6781.500000	3196.000000	7.340946e+06
max	50.000000	109008.000000	52070.000000	3.946159e+07

Parts of a DataFrame

To better understand DataFrame objects, it's useful to know that they consist of three components, stored as attributes:

.values: A two-dimensional NumPy array of values.

.columns: An index of columns: the column names.

.index: An index for the rows: either row numbers or row names.

You can usually think of indexes as a list of strings or numbers, though the pandas Index data type allows for more sophisticated options. (These will be covered later in the course.)

TASKS

Print a 2D NumPy array of the values in homelessness.

Print the column names of homelessness.

Print the index of homelessness

In [25]:

```
# Print the values of homelessness
print(homelessness.values)

# Print the column index of homelessness
print(homelessness.columns)

# Print the row index of homelessness
print(homelessness.index)
```

```
[[0 'East South Central' 'Alabama' 2570.0 864.0 4887681]
 [1 'Pacific' 'Alaska' 1434.0 582.0 735139]
 [2 'Mountain' 'Arizona' 7259.0 2606.0 7158024]
 [3 'West South Central' 'Arkansas' 2280.0 432.0 3009733]
 [4 'Pacific' 'California' 109008.0 20964.0 39461588]
 [5 'Mountain' 'Colorado' 7607.0 3250.0 5691287]
 [6 'New England' 'Connecticut' 2280.0 1696.0 3571520]
 [7 'South Atlantic' 'Delaware' 708.0 374.0 965479]
 [8 'South Atlantic' 'District of Columbia' 3770.0 3134.0 701547]
 [9 'South Atlantic' 'Florida' 21443.0 9587.0 21244317]
 [10 'South Atlantic' 'Georgia' 6943.0 2556.0 10511131]
 [11 'Pacific' 'Hawaii' 4131.0 2399.0 1420593]
 [12 'Mountain' 'Idaho' 1297.0 715.0 1750536]
 [13 'East North Central' 'Illinois' 6752.0 3891.0 12723071]
 [14 'West North Central' 'Indiana' 2776.0 1400.0 6605407]
 [15 'West North Central' 'Iowa' 1875.0 1000.0 3190714]
 [16 'West North Central' 'Kansas' 1752.0 896.0 3595129]
 [17 'Mountain' 'Kentucky' 2069.0 1100.0 4477943]
 [18 'West South Central' 'Louisiana' 1485.0 784.0 4653767]
 [19 'West South Central' 'Maine' 577.0 294.0 1359709]
 [20 'New England' 'Maryland' 2712.0 1400.0 6065407]
 [21 'West South Central' 'Massachusetts' 859.0 444.0 1782179]
 [22 'New England' 'Michigan' 9399.0 5000.0 10717361]
 [23 'West North Central' 'Minnesota' 5629.0 2995.0 5629000]
 [24 'West North Central' 'Mississippi' 2967.0 1554.0 2967000]
 [25 'West North Central' 'Missouri' 5937.0 3114.0 5937000]
 [26 'West North Central' 'Montana' 1074.0 562.0 1074000]
 [27 'Mountain' 'Nebraska' 1961.0 1036.0 1961000]
 [28 'West North Central' 'Nevada' 2069.0 1100.0 2069000]
 [29 'Mountain' 'New Hampshire' 623.0 324.0 623000]
 [30 'New England' 'New Jersey' 8929.0 4714.0 8929000]
 [31 'New England' 'New Mexico' 1961.0 1036.0 1961000]
 [32 'West South Central' 'New York' 1961.0 1036.0 1961000]
 [33 'New England' 'North Carolina' 5629.0 2995.0 5629000]
 [34 'West South Central' 'North Dakota' 623.0 324.0 623000]
 [35 'Mountain' 'Ohio' 9399.0 5000.0 9399000]
 [36 'West North Central' 'Oklahoma' 1961.0 1036.0 1961000]
 [37 'Mountain' 'Oregon' 1074.0 562.0 1074000]
 [38 'Mountain' 'Pennsylvania' 9399.0 5000.0 9399000]
 [39 'New England' 'Rhode Island' 623.0 324.0 623000]
 [40 'New England' 'South Carolina' 2967.0 1554.0 2967000]
 [41 'West South Central' 'South Dakota' 623.0 324.0 623000]
 [42 'Mountain' 'Tennessee' 7607.0 3250.0 7607000]
 [43 'West South Central' 'Texas' 109008.0 20964.0 109008000]
 [44 'Mountain' 'Utah' 7259.0 2606.0 7259000]
 [45 'Mountain' 'Vermont' 623.0 324.0 623000]
 [46 'New England' 'Virginia' 2280.0 432.0 2280000]
 [47 'South Atlantic' 'Washington' 7259.0 2606.0 7259000]
 [48 'Mountain' 'West Virginia' 1074.0 562.0 1074000]
 [49 'West South Central' 'Wisconsin' 5629.0 2995.0 5629000]
 [50 'West North Central' 'Wyoming' 1074.0 562.0 1074000]]
```

```
[14 'East North Central' 'Indiana' 3776.0 1482.0 669549]]
[15 'West North Central' 'Iowa' 1711.0 1038.0 3148618]
[16 'West North Central' 'Kansas' 1443.0 773.0 2911359]
[17 'East South Central' 'Kentucky' 2735.0 953.0 4461153]
[18 'West South Central' 'Louisiana' 2540.0 519.0 4659690]
[19 'New England' 'Maine' 1450.0 1066.0 1339057]
[20 'South Atlantic' 'Maryland' 4914.0 2230.0 6035802]
[21 'New England' 'Massachusetts' 6811.0 13257.0 6882635]
[22 'East North Central' 'Michigan' 5209.0 3142.0 9984072]
[23 'West North Central' 'Minnesota' 3993.0 3250.0 5606249]
[24 'East South Central' 'Mississippi' 1024.0 328.0 2981020]
[25 'West North Central' 'Missouri' 3776.0 2107.0 6121623]
[26 'Mountain' 'Montana' 983.0 422.0 1060665]
[27 'West North Central' 'Nebraska' 1745.0 676.0 1925614]
[28 'Mountain' 'Nevada' 7058.0 486.0 3027341]
[29 'New England' 'New Hampshire' 835.0 615.0 1353465]
[30 'Mid-Atlantic' 'New Jersey' 6048.0 3350.0 8886025]
[31 'Mountain' 'New Mexico' 1949.0 602.0 2092741]
[32 'Mid-Atlantic' 'New York' 39827.0 52070.0 19530351]
[33 'South Atlantic' 'North Carolina' 6451.0 2817.0 10381615]
[34 'West North Central' 'North Dakota' 467.0 75.0 758080]
[35 'East North Central' 'Ohio' 6929.0 3320.0 11676341]
[36 'West South Central' 'Oklahoma' 2823.0 1048.0 3940235]
[37 'Pacific' 'Oregon' 11139.0 3337.0 4181886]
[38 'Mid-Atlantic' 'Pennsylvania' 8163.0 5349.0 12800922]
[39 'New England' 'Rhode Island' 747.0 354.0 1058287]
[40 'South Atlantic' 'South Carolina' 3082.0 851.0 5084156]
[41 'West North Central' 'South Dakota' 836.0 323.0 878698]
[42 'East South Central' 'Tennessee' 6139.0 1744.0 6771631]
[43 'West South Central' 'Texas' 19199.0 6111.0 28628666]
[44 'Mountain' 'Utah' 1904.0 972.0 3153550]
[45 'New England' 'Vermont' 780.0 511.0 624358]
[46 'South Atlantic' 'Virginia' 3928.0 2047.0 8501286]
[47 'Pacific' 'Washington' 16424.0 5880.0 7523869]
[48 'South Atlantic' 'West Virginia' 1021.0 222.0 1804291]
[49 'East North Central' 'Wisconsin' 2740.0 2167.0 5807406]
[50 'Mountain' 'Wyoming' 434.0 205.0 577601]]
Index(['Unnamed: 0', 'region', 'state', 'individuals', 'family_members',
      'state_pop'],
      dtype='object')
RangeIndex(start=0, stop=51, step=1)
```

SORTING ROWS

Finding interesting bits of data in a DataFrame is often easier if you change the order of the rows. You can sort the rows by passing a column name to `.sort_values()`.

In cases where rows have the same value (this is common if you sort on a categorical variable), you may wish to break the ties by sorting on another column. You can sort on multiple columns in this way by passing a list of column names.

Sort on ... Syntax

one column `df.sort_values("breed")`

multiple columns `df.sort_values(["breed", "weight_kg"])`

By combining `.sort_values()` with `.head()`, you can answer questions in the form, "What are the top cases where...?".

TASK

Sort homelessness by the number of homeless individuals, from smallest to largest, and save this as *homelessness_ind*.

Print the head of the sorted DataFrame.

In [26]:

```
# Sort homelessness by individual
homelessness_ind = homelessness.sort_values("individuals")

# Print the top few rows
print(homelessness_ind.head())
```

	Unnamed: 0	region	state	individuals	family_members	\
50	50	Mountain	Wyoming	434.0	205.0	
34	34	West North Central	North Dakota	467.0	75.0	
7	7	South Atlantic	Delaware	708.0	374.0	
39	39	New England	Rhode Island	747.0	354.0	
45	45	New England	Vermont	780.0	511.0	

	state_pop
50	577601
34	758080
7	965479
39	1058287
45	624358

TASK

Sort homelessness by the number of homeless family_members in descending order, and save this as homelessness_fam.

Print the head of the sorted DataFrame.

In [27]:

```
# Sort homelessness by descending family members
homelessness_fam = homelessness.sort_values("family_members", ascending = False)

# Print the top few rows
print(homelessness_fam.head())
```

	Unnamed: 0	region	state	individuals	\
32	32	Mid-Atlantic	New York	39827.0	
4	4	Pacific	California	109008.0	
21	21	New England	Massachusetts	6811.0	
9	9	South Atlantic	Florida	21443.0	
43	43	West South Central	Texas	19199.0	

	family_members	state_pop
32	52070.0	19530351
4	20964.0	39461588
21	13257.0	6882635
9	9587.0	21244317
43	6111.0	28628666

Sort homelessness first by region (ascending), and then by number of family members (descending). Save this as homelessness_reg_fam. Print the head of the sorted DataFrame.

In [28]:

```
# Sort homelessness by region, then descending family members
homelessness_reg_fam = homelessness.sort_values(["region", "family_members"], ascending = [True, False])

# Print the top few rows
print(homelessness_reg_fam.head())
```

	Unnamed: 0	region	state	individuals	family_members	\
10	10	Mid-Atlantic	New York	39827.0	52070.0	

13	13	East North Central	Illinois	6752.0	3891.0
35	35	East North Central	Ohio	6929.0	3320.0
22	22	East North Central	Michigan	5209.0	3142.0
49	49	East North Central	Wisconsin	2740.0	2167.0
14	14	East North Central	Indiana	3776.0	1482.0

	state_pop
13	12723071
35	11676341
22	9984072
49	5807406
14	6695497

Subsetting columns

When working with data, you may not need all of the variables in your dataset.

Square brackets (`[]`) can be used to select only the columns that matter to you in an order that makes sense to you.

To select only "col_a" of the DataFrame `df`, use `df["col_a"]`

To select "col_a" and "col_b" of `df`, use `df[["col_a", "col_b"]]`

TAST Create a DataFrame called `individuals` that contains only the individuals column of `homelessness`.

Print the head of the result.

In [29]:

```
# Select the individuals column
individuals = homelessness["individuals"]

# Print the head of the result
print(individuals.head())
```

```
0      2570.0
1      1434.0
2      7259.0
3      2280.0
4    109008.0
Name: individuals, dtype: float64
```

Create a DataFrame called `state_fam` that contains only the state and *family_members* columns of *homelessness*, in that order.

Print the head of the result.

In [30]:

```
# Select the state and family_members columns
state_fam = homelessness[["state", "family_members"]]

# Print the head of the result
print(state_fam.head())
```

```
      state  family_members
0  Alabama          864.0
1  Alaska           582.0
2  Arizona         2606.0
3  Arkansas          432.0
4  California       20964.0
```

Create a DataFrame called `ind_state` that contains the individuals and state columns of *homelessness*, in that order.

Print the head of the result.

In [31]:

```
# Select only the individuals and state columns, in that order
ind_state = homelessness[["individuals", "state"]]

# Print the head of the result
print(ind_state.head())
```

	individuals	state
0	2570.0	Alabama
1	1434.0	Alaska
2	7259.0	Arizona
3	2280.0	Arkansas
4	109008.0	California

Subsetting rows

A large part of data science is about finding which bits of your dataset are interesting. One of the simplest techniques for this is to find a subset of rows that match some criteria. This is sometimes known as filtering rows or selecting rows.

There are many ways to subset a DataFrame, perhaps the most common is to use relational operators to return True or False for each row, then pass that inside square brackets.

```
dogs[dogs["height_cm"] > 60]
dogs[dogs["color"] == "tan"]
```

You can filter for multiple conditions at once by using the *"bitwise and" operator, &*.

```
dogs[(dogs["height_cm"] > 60) & (dogs["color"] == "tan")]
```

...

Tasks

Filter homelessness for cases where the number of individuals is greater than ten thousand, assigning to ind_gt_10k.

View the printed result.

In [32]:

```
# Filter for rows where individuals is greater than 10000
ind_gt_10k = homelessness[homelessness["individuals"]>10000]

# See the result
print(ind_gt_10k)
```

Unnamed: 0		region	state	individuals	family_members	\
4	4	Pacific	California	109008.0	20964.0	
9	9	South Atlantic	Florida	21443.0	9587.0	
32	32	Mid-Atlantic	New York	39827.0	52070.0	
37	37	Pacific	Oregon	11139.0	3337.0	
43	43	West South Central	Texas	19199.0	6111.0	
47	47	Pacific	Washington	16424.0	5880.0	

	state_pop
4	39461588
9	21244317
32	19530351
37	4181886
43	28628666
47	7523869

Filter homelessness for cases where the USA Census region is "Mountain", assigning to mountain_reg. View the printed result.

In [33]:

```
# Filter for rows where region is Mountain
mountain_reg = homelessness[homelessness['region']=='Mountain']

# See the result
print(mountain_reg)
```

	Unnamed: 0	region	state	individuals	family_members	state_pop
2	2	Mountain	Arizona	7259.0	2606.0	7158024
5	5	Mountain	Colorado	7607.0	3250.0	5691287
12	12	Mountain	Idaho	1297.0	715.0	1750536
26	26	Mountain	Montana	983.0	422.0	1060665
28	28	Mountain	Nevada	7058.0	486.0	3027341
31	31	Mountain	New Mexico	1949.0	602.0	2092741
44	44	Mountain	Utah	1904.0	972.0	3153550
50	50	Mountain	Wyoming	434.0	205.0	577601

Filter homelessness for cases where the number of family_members is less than one thousand and the region is "Pacific", assigning to fam_lt_1k_pac. View the printed result.

In [34]:

```
# Filter for rows where family_members is less than 1000
# and region is Pacific
fam_lt_1k_pac = homelessness[(homelessness['family_members']<1000) & (homelessness['region']=='Pacific')]

# See the result
print(fam_lt_1k_pac)
```

	Unnamed: 0	region	state	individuals	family_members	state_pop
1	1	Pacific	Alaska	1434.0	582.0	735139

Subsetting rows by categorical variables

Subsetting data based on a categorical variable often involves using the "or" operator (|) to select rows from multiple categories.

This can get tedious when you want all states in one of three different regions, for example.

Instead, use the .isin() method, which will allow you to tackle this problem by writing one condition instead of three separate ones.

```
colors = ["brown", "black", "tan"]
condition = dogs["color"].isin(colors)
dogs[condition]
```

...

Filter homelessness for cases where the USA census region is "South Atlantic" or it is "Mid-Atlantic", assigning to south_mid_atlantic.

View the printed result.

In [35]:

```
# Subset for rows in South Atlantic or Mid-Atlantic regions
li = ["South Atlantic", "Mid-Atlantic"]
south_mid_atlantic = homelessness[homelessness['region'].isin(li)]

# See the result
print(south_mid_atlantic)
```

	Unnamed: 0	region	state	individuals	\
7	7	South Atlantic	Delaware	708.0	
8	8	South Atlantic	District of Columbia	3770.0	
9	9	South Atlantic	Florida	21443.0	
10	10	South Atlantic	Georgia	6943.0	
20	20	South Atlantic	Maryland	4914.0	
30	30	Mid-Atlantic	New Jersey	6048.0	
32	32	Mid-Atlantic	New York	39827.0	
33	33	South Atlantic	North Carolina	6451.0	

35	35	South Atlantic	North Carolina	6451.0
38	38	Mid-Atlantic	Pennsylvania	8163.0
40	40	South Atlantic	South Carolina	3082.0
46	46	South Atlantic	Virginia	3928.0
48	48	South Atlantic	West Virginia	1021.0

	family_members	state_pop
7	374.0	965479
8	3134.0	701547
9	9587.0	21244317
10	2556.0	10511131
20	2230.0	6035802
30	3350.0	8886025
32	52070.0	19530351
33	2817.0	10381615
38	5349.0	12800922
40	851.0	5084156
46	2047.0	8501286
48	222.0	1804291

Filter homelessness for cases where the USA census state is in the list of Mojave states, canu, assigning to `mojave_homelessness`.

View the printed result.

In [36]:

```
# The Mojave Desert states
canu = ["California", "Arizona", "Nevada", "Utah"]

# Filter for rows in the Mojave Desert states
mojave_homelessness = homelessness[homelessness["state"].isin(canu)]

# See the result
print(mojave_homelessness)
```

	Unnamed: 0	region	state	individuals	family_members	state_pop
2	2	Mountain	Arizona	7259.0	2606.0	7158024
4	4	Pacific	California	109008.0	20964.0	39461588
28	28	Mountain	Nevada	7058.0	486.0	3027341
44	44	Mountain	Utah	1904.0	972.0	3153550

Adding new columns

You aren't stuck with just the data you are given. Instead, you can add new columns to a DataFrame. This has many names, such as transforming, mutating, and feature engineering.

You can create new columns from scratch, but it is also common to derive them from other columns, for example, by adding columns together or by changing their units.

...

Add a new column to `homelessness`, named `total`, containing the sum of the `individuals` and `family_members` columns.

Add another column to `homelessness`, named `p_individuals`, containing the proportion of homeless people in each state who are individuals

In [37]:

```
# Add total col as sum of individuals and family_members
homelessness['total'] = homelessness['individuals'] + homelessness['family_members']

# Add p_individuals col as proportion of individuals
homelessness['p_individuals'] = homelessness['individuals']/homelessness['total']

# See the result
print(homelessness)
```

	Unnamed: 0	region	state	individuals	family_members	total	p_individuals
--	------------	--------	-------	-------------	----------------	-------	---------------

0		East South Central	Alabama	2570.0
1	1	Pacific	Alaska	1434.0
2	2	Mountain	Arizona	7259.0
3	3	West South Central	Arkansas	2280.0
4	4	Pacific	California	109008.0
5	5	Mountain	Colorado	7607.0
6	6	New England	Connecticut	2280.0
7	7	South Atlantic	Delaware	708.0
8	8	South Atlantic	District of Columbia	3770.0
9	9	South Atlantic	Florida	21443.0
10	10	South Atlantic	Georgia	6943.0
11	11	Pacific	Hawaii	4131.0
12	12	Mountain	Idaho	1297.0
13	13	East North Central	Illinois	6752.0
14	14	East North Central	Indiana	3776.0
15	15	West North Central	Iowa	1711.0
16	16	West North Central	Kansas	1443.0
17	17	East South Central	Kentucky	2735.0
18	18	West South Central	Louisiana	2540.0
19	19	New England	Maine	1450.0
20	20	South Atlantic	Maryland	4914.0
21	21	New England	Massachusetts	6811.0
22	22	East North Central	Michigan	5209.0
23	23	West North Central	Minnesota	3993.0
24	24	East South Central	Mississippi	1024.0
25	25	West North Central	Missouri	3776.0
26	26	Mountain	Montana	983.0
27	27	West North Central	Nebraska	1745.0
28	28	Mountain	Nevada	7058.0
29	29	New England	New Hampshire	835.0
30	30	Mid-Atlantic	New Jersey	6048.0
31	31	Mountain	New Mexico	1949.0
32	32	Mid-Atlantic	New York	39827.0
33	33	South Atlantic	North Carolina	6451.0
34	34	West North Central	North Dakota	467.0
35	35	East North Central	Ohio	6929.0
36	36	West South Central	Oklahoma	2823.0
37	37	Pacific	Oregon	11139.0
38	38	Mid-Atlantic	Pennsylvania	8163.0
39	39	New England	Rhode Island	747.0
40	40	South Atlantic	South Carolina	3082.0
41	41	West North Central	South Dakota	836.0
42	42	East South Central	Tennessee	6139.0
43	43	West South Central	Texas	19199.0
44	44	Mountain	Utah	1904.0
45	45	New England	Vermont	780.0
46	46	South Atlantic	Virginia	3928.0
47	47	Pacific	Washington	16424.0
48	48	South Atlantic	West Virginia	1021.0
49	49	East North Central	Wisconsin	2740.0
50	50	Mountain	Wyoming	434.0

	family_members	state_pop	total	p_individuals
0	864.0	4887681	3434.0	0.748398
1	582.0	735139	2016.0	0.711310
2	2606.0	7158024	9865.0	0.735834
3	432.0	3009733	2712.0	0.840708
4	20964.0	39461588	129972.0	0.838704
5	3250.0	5691287	10857.0	0.700654
6	1696.0	3571520	3976.0	0.573441
7	374.0	965479	1082.0	0.654344
8	3134.0	701547	6904.0	0.546060
9	9587.0	21244317	31030.0	0.691041
10	2556.0	10511131	9499.0	0.730919
11	2399.0	1420593	6530.0	0.632619
12	715.0	1750536	2012.0	0.644632
13	3891.0	12723071	10643.0	0.634408
14	1482.0	6695497	5258.0	0.718144
15	1038.0	3148618	2749.0	0.622408
16	773.0	2911359	2216.0	0.651173
17	953.0	4461153	3688.0	0.741594
18	519.0	4659690	3059.0	0.830337

19	1066.0	1339057	2516.0	0.576312
20	2230.0	6035802	7144.0	0.687850
21	13257.0	6882635	20068.0	0.339396
22	3142.0	9984072	8351.0	0.623758
23	3250.0	5606249	7243.0	0.551291
24	328.0	2981020	1352.0	0.757396
25	2107.0	6121623	5883.0	0.641849
26	422.0	1060665	1405.0	0.699644
27	676.0	1925614	2421.0	0.720777
28	486.0	3027341	7544.0	0.935578
29	615.0	1353465	1450.0	0.575862
30	3350.0	8886025	9398.0	0.643541
31	602.0	2092741	2551.0	0.764014
32	52070.0	19530351	91897.0	0.433387
33	2817.0	10381615	9268.0	0.696051
34	75.0	758080	542.0	0.861624
35	3320.0	11676341	10249.0	0.676066
36	1048.0	3940235	3871.0	0.729269
37	3337.0	4181886	14476.0	0.769481
38	5349.0	12800922	13512.0	0.604130
39	354.0	1058287	1101.0	0.678474
40	851.0	5084156	3933.0	0.783626
41	323.0	878698	1159.0	0.721311
42	1744.0	6771631	7883.0	0.778764
43	6111.0	28628666	25310.0	0.758554
44	972.0	3153550	2876.0	0.662031
45	511.0	624358	1291.0	0.604183
46	2047.0	8501286	5975.0	0.657406
47	5880.0	7523869	22304.0	0.736370
48	222.0	1804291	1243.0	0.821400
49	2167.0	5807406	4907.0	0.558386
50	205.0	577601	639.0	0.679186

Combo-attack! You've seen the four most common types of data manipulation: sorting rows, subsetting columns, subsetting rows, and adding new columns.

In a real-life data analysis, you can mix and match these four manipulations to answer a multitude of questions.

In this exercise, you'll answer the question, "Which state has the highest number of homeless individuals per 10,000 people in the state?" Combine your new pandas skills to find out.

Add a column to homelessness, indiv_per_10k, containing the number of homeless individuals per ten thousand people in each state.

Subset rows where indiv_per_10k is higher than 20, assigning to high_homelessness.

Sort high_homelessness by descending indiv_per_10k, assigning to high_homelessness_srt.

Select only the state and indiv_per_10k columns of high_homelessness_srt and save as result. Look at the result.

...

In [38]:

```
# Create indiv_per_10k col as homeless individuals per 10k state pop
homelessness["indiv_per_10k"] = 10000 * (homelessness['individuals'] /homelessness['state_pop'])

# Subset rows for indiv_per_10k greater than 20
high_homelessness = homelessness[homelessness['indiv_per_10k']>20]

# Sort high_homelessness by descending indiv_per_10k
high_homelessness_srt = high_homelessness.sort_values('indiv_per_10k',ascending=False)

# From high_homelessness_srt, select the state and indiv_per_10k cols
result = high_homelessness_srt[['state', 'indiv_per_10k']]

# See the result
print(result)
```

state indiv per 10k

8	District of Columbia	53.738381
11	Hawaii	29.079406
4	California	27.623825
37	Oregon	26.636307
28	Nevada	23.314189
47	Washington	21.829195
32	New York	20.392363