Nested JSON Responses

Module 2 | Chapter 5 | Notebook 8

In this lesson you will look at nested JSON objects. These are shown as nested dictionaries in Python, i.e., dictionaries within dictionaries. By the end of this exercise you will be able to:

- Access a nested JSON file
- Visualize geographic clusters with a scatter plot

Accessing nested data

Scenario: After you found out which countries visitors to the company website came from, the marketing department approached you to ask whether you can also find out which cities most visitors were based in and which region in the world most visitors came from.

In order to complete this task you will need to access an API again. Start by importing the `requests` module (without an alias).

In [128…
```
import requests
```

There is a third API that you haven't used yet. It is a combination of the IP API from *API Requests* and the user_agent API from *JSON Responses*. You can request geographical information from the IP addresses and system information from the user agent in a single API request. The API's URL is:

- *http://api.stackfuel.com:5000/all_info*

You will need a different authentication token this time:

```
'data-analysts-want-all-json'
```

Now you have to specify two API request parameters to indicate which input is an IP address and which input is a user agent:

- `'ip_address'`
- `'user_agent'`

A request to this API would look something like this:

In [129…
```
response = requests.get('http://api.stackfuel.com:5000/all_info',
                        params={'user_agent': 'Mozilla/5.0 (X11; Trisquel; Linux x86
                                'ip_address': '208.118.235.148'},
                        headers={"Authorization": 'data-analysts-want-all-json'})
```

The API returns a JSON object. Store the JSON object from the last API request as a `dict` in the new variable `response_dict` and then print this.

In [130…

```
response_dict = response.json()
response_dict
```

Out[130…
```
{'browser': {'name': 'GNU IceCat', 'version': '52.6.0'},
 'platform': {'name': 'Trisquel'},
 'device': {'name': 'GNU/Linux Desktop', 'type': 'Desktop'},
 'location': {'country': {'name': 'United States'},
  'city': {'name': 'Boston',
   'geo_location': {'longitude': -71.0598, 'latitude': 42.3584}}}}
```

Now you will see that there are a lot of curly brackets `{}` enclosed inside other curly brackets. These are dictionaries within other dictionaries. The `dict`

```
{'name': 'Safari', 'version': '11.0'}
```

is simply the value under the key `'browser'` in the higher-level dictionary. To access this `dict` you need the key `'browser'`. You can then store the result in `response_dict_browser`.

In [131…
```
response_dict_browser = response_dict['browser']
response_dict_browser
```

Out[131…
```
{'name': 'GNU IceCat', 'version': '52.6.0'}
```

`response_dict_browser` is now the `dict` that is contained under the key `'browser'`. You can access it like you would a normal `dict`. This is how you would access the value under the key `'version'`.

In [132…
```
response_dict_browser_version = response_dict_browser['version']
response_dict_browser_version
```

Out[132…
```
'52.6.0'
```

You don't necessarily have to store the result of the nested dictionary in a variable to access the entry in the nested dictionary. You can also store the browser version in `response_dict_browser_version` straight away without creating the variable `response_dict_browser`. To do this, you write the keys that you use to access each level one after the other. You start with the highest-level key (in this case `'browser'`) and then work your way down step by step (in this case: `'version'`). This would look something like this:

In [133…
```
response_dict_browser_version = response_dict['browser']['name']
response_dict_browser_version
```

Out[133…
```
'GNU IceCat'
```

Now you know how to access entries in nested dictionaries. The same principle applies when there are three levels of nesting, for example under the `'location'` key. If you were storing the intermediate steps in separate variables, you would access the latitude value (`'latitude'`) of the website visitors as follows:

In [134…
```
response_dict_location = response_dict['location']
print(response_dict_location, '\n')

response_dict_location_city = response_dict_location['city']
```

```
print(response_dict_location_city, '\n')

response_dict_location_city_geo_location = response_dict_location_city['geo_location
print(response_dict_location_city_geo_location, '\n')

response_dict_location_city_geo_location_latitude = response_dict_location_city_geo_
print(response_dict_location_city_geo_location_latitude)
```

```
{'country': {'name': 'United States'}, 'city': {'name': 'Boston', 'geo_location':
{'longitude': -71.0598, 'latitude': 42.3584}}}

{'name': 'Boston', 'geo_location': {'longitude': -71.0598, 'latitude': 42.3584}}

{'longitude': -71.0598, 'latitude': 42.3584}

42.3584
```

Without storing the intermediate steps in different variables, this is how you would access the same information. You write the keys that you use to access each level one after the other.

In [135…
```
response_dict_location_city_geo_location_latitude = response_dict['location']['city'
response_dict_location_city_geo_location_latitude
```

Out[135… `42.3584`

Now try it out for yourself. You can look at the `dict` again to remind yourself how it's structured:

In [136…
```
response_dict_location_city_geo_location_longitude = response_dict['location']['city
response_dict_location_city_geo_location_longitude

response_dict_location_city_name = response_dict['location']['city']['name']
response_dict_location_city_name
```

Out[136… `'Boston'`

Save the longitude (`'longitude'`) in the new variable `respone_dict_location_city_geo_location_longitude` . The longitude is stored under the key `'geo_location'` , which is stored under `'city'` which you can find under `'location'` .

In [137…
```
response_dict_location_city_geo_location_longitude = response_dict['location']['city
response_dict_location_city_geo_location_longitude
```

Out[137… `-71.0598`

Which city are these coordinates located in? Store the city name in the new variable `response_dict_location_city_name` .

In [138…
```
response_dict_location_city_name = response_dict['location']['city']['name']
response_dict_location_city_name
```

Out[138… `'Boston'`

Which country are these coordinates located in? Save the country name in `response_dict_location_country` .

In [139…
```
response_dict_location_country = response_dict['location']['country']
response_dict_location_country
```

Out[139…
```
{'name': 'United States'}
```

Congratulations: You have learned how to access a nested JSON file. Now you can apply what you have learned to store the information you need in a `DataFrame`. Then you will be able to analyze the data.

Accessing a dict to populate a `DataFrame`

You have already processed the data you need in the last exercise. It's stored in the file *request_ips_user_agents_5k.csv*, which is located in the current working directory. Store the data in a `DataFrame` called `df`. Then print the beginning of `df`.

Tip: First import `pandas` with its conventional alias `pd`. Specify `'|'` as the quote character (`quotechar`).

In [140…
```
import pandas as pd
df = pd.read_csv('request_ips_user_agents_5k.csv',sep = ',', quotechar = '|')
df.head(10)
```

Out[140…

	ip	user_agent
0	216.216.56.64	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...
1	189.109.167.60	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...
2	249.136.57.232	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...
3	10.7.64.175	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...
4	222.4.129.210	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...
5	32.195.5.171	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...
6	181.95.62.227	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...
7	56.219.74.171	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...
8	205.159.70.242	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...
9	211.94.3.180	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...

Add the following empty columns to `df`:

- `'country'`
- `'city'`
- `'longitude'`
- `'latitude'`

In [141…
```
df.loc[:,'country'] = ''
df.loc[:,'city'] = ''
df.loc[:,'longitude'] = ''
df.loc[:,'latitude'] = ''


df
```

Out[141…

	ip	user_agent	country	city	longitude	latitude
0	216.216.56.64	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...				
1	189.109.167.60	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...				
2	249.136.57.232	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...				
3	10.7.64.175	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...				
4	222.4.129.210	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...				
...	...	...	...	...	...	...
4995	246.200.179.75	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...				
4996	212.44.76.212	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...				
4997	77.102.209.233	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...				
4998	7.30.45.77	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...				
4999	25.189.232.162	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...				

5000 rows × 6 columns

Let's focus on the first row to start with. Formulate an API query to *http://api.stackfuel.com:5000/all_info* with the IP address and the user agent from the first line of `df`. Save the data that the API sends back in the variable `response`. Use the `my_response.json()` method to convert the data in the JSON object into a `dict`. Store this `dict` in the variable `response_dict`.

In [142…

```
import time
for i in range(df.shape[0]):
    location_details = df.iloc[i,1]
    ip_address = df.iloc[i,0]
    api_url = 'http://api.stackfuel.com:5000/user_agent/json'
    response = requests.get('http://api.stackfuel.com:5000/all_info',
                            params={'user_agent': "{}".format(location_details),
                                    'ip_address': "{}".format(ip_address)},
                            headers={"Authorization": 'data-analysts-want-all-json'})
    if (i%100 == 0):
        time.sleep(2)
    response_dict = response.json()
    df.iloc[i,2] = response_dict['location']['country']['name']
    df.iloc[i,3] = response_dict['location']['city']['name']
    df.iloc[i,4] = response_dict['location']['city']['geo_location']['longitude']
    df.iloc[i,5] = response_dict['location']['city']['geo_location']['latitude']
```

You can then access the relevant information from this `dict` and store it in `df`. For example, you could access the country names as follows and store them in the `'country'` column of

`df` as follows:

In [143…

```
df.head(30)
```

Out[143…

	ip	user_agent	country	city	longitude	latitude
0	216.216.56.64	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	United States	None	-97.822	37.751
1	189.109.167.60	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	Brazil	None	-43.2192	-22.8305
2	249.136.57.232	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	unknown	unknown	unknown	unknown
3	10.7.64.175	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	unknown	unknown	unknown	unknown
4	222.4.129.210	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	Japan	Tokyo	139.751	35.685
5	32.195.5.171	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	United States	None	-97.822	37.751
6	181.95.62.227	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	Argentina	Córdoba	-64.1611	-31.2389
7	56.219.74.171	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	United States	Raleigh	-78.6253	35.7977
8	205.159.70.242	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	United States	New York	-73.9885	40.7317
9	211.94.3.180	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	China	None	113.727	34.7725
10	107.197.44.42	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	United States	Chicago	-87.6368	41.8998
11	211.46.208.64	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	Republic of Korea	Sejong	127.292	36.593
12	62.1.126.114	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	Greece	None	23.7167	37.9667
13	157.171.197.86	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	Sweden	Gothenburg	11.9667	57.7167
14	249.26.135.174	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	unknown	unknown	unknown	unknown
15	143.154.218.199	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	United States	Montgomery	-86.2539	32.404
16	110.180.26.140	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59....	China	Xi'an	108.929	34.2583
17	236.24.118.99	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59....	unknown	unknown	unknown	unknown
18	252.191.102.202	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	unknown	unknown	unknown	unknown
19	145.65.63.150	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	Netherlands	Arnhem	5.9256	51.9798
20	196.160.30.37	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	South Africa	None	24	-29

	ip	user_agent	country	city	longitude	latitude
21	165.15.12.23	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	unknown	unknown	unknown	unknown
22	247.251.169.118	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	unknown	unknown	unknown	unknown
23	63.35.156.180	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	United States	Seattle	-122.312	47.542
24	70.3.73.65	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	United States	None	-97.822	37.751
25	167.237.114.139	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	United States	Minneapolis	-93.3343	44.8743
26	224.61.177.138	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	unknown	unknown	unknown	unknown
27	107.73.29.126	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	United States	None	-97.822	37.751
28	210.244.127.142	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	Taiwan	Tainan City	120.213	22.9908
29	149.110.11.145	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59....	United States	None	-97.822	37.751

Now save the city names, latitude, and longitude of this row in the relevant columns of `df`. Then print the beginning of `df`.

In [144...

```
df
```

Out[144...

	ip	user_agent	country	city	longitude	latitude
0	216.216.56.64	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	United States	None	-97.822	37.751
1	189.109.167.60	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	Brazil	None	-43.2192	-22.8305
2	249.136.57.232	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	unknown	unknown	unknown	unknown
3	10.7.64.175	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	unknown	unknown	unknown	unknown
4	222.4.129.210	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	Japan	Tokyo	139.751	35.685
...	...	...	...	...	...	...
4995	246.200.179.75	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	unknown	unknown	unknown	unknown
4996	212.44.76.212	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	Russia	Kaliningrad	20.511	54.7065
4997	77.102.209.233	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	United Kingdom	Preston	-2.7167	53.7667
4998	7.30.45.77	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	United States	None	-97.822	37.751
4999	25.189.232.162	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	United Kingdom	None	-0.1224	51.4964

5000 rows × 6 columns

Great! You have processed the first row. Now you can write a `for` loop to iterate through all the rows in `df` and populate the four new columns on the right with information. Use the API *http://api.stackfuel.com:5000/all_info* with the authorization token `'data-analysts-want-all-json'`. At each loop iteration you will receive a JSON object from the API. Convert this into a `dict`, which you can then access to fill in the empty columns of `df` correctly.

Tip: Remember that this API doesn't allow more than 100 requests per second. First import the `time` module and use the `time.sleep()` function.

In [ ]:
```
```

Now we have all the information we need, so we should close the connection to `response` in the following cell.

In [145…
```
response.close()
```

Congratulations: You have stored the coordinates in a data structure that you can analyze easily: `DataFrame`. Now you can start analyzing the data.

Analyzing coordinates

Print the beginning of `df`, in order to get a first impression of the data.

In [146…
```
df.head()
```

Out[146…

	ip	user_agent	country	city	longitude	latitude
0	216.216.56.64	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	United States	None	-97.822	37.751
1	189.109.167.60	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	Brazil	None	-43.2192	-22.8305
2	249.136.57.232	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	unknown	unknown	unknown	unknown
3	10.7.64.175	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	unknown	unknown	unknown	unknown
4	222.4.129.210	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	Japan	Tokyo	139.751	35.685

Your task now is to find out which cities the most website visitors come from and which region in the world the most visitors come from. Let's start with the most common cities. Which ten cities were the most common?Which category is the most frequent?

Count how often different cities in the `'city'` column occur. Save the number in a `DataFrame` called `df_city_count`, with the city names as row names and a single column indicating how often this city occurs in `df`. Then print the first 20 rows of all the columns in `df`.

In [147…
```
df_city_count = pd.crosstab(index= df.loc[:,'city'], columns = 'count')
```

```
df_city_count.head(20)
df_city_count.shape
#df.groupby('city').agg('count').sort_values(by= 'ip',ascending =False)
```

Out[147… (1221, 1)

You can use the `my_df.sort_values()` method to sort the cities by frequency (see *Querying an API*). Assign the column name you want to sort the values by to the `by` parameter. You also want to sort the data from the largest value to the smallest value, rather than the default setting of smallest to largest (`ascending=False`). Then print the first 20 rows of all the columns in `df_city_count` .

In [148…

```
sorted_countwise = df_city_count.sort_values(by = 'count',ascending = False)
sorted_countwise
```

Out[148…

col_0	count
city	
unknown	762
Beijing	84
Fort Huachuca	69
Tokyo	65
Palo Alto	39
...	...
Hualien City	1
Huelma	1
Huilong	1
Humble	1
Šiauliai	1

1221 rows × 1 columns

Now you can visualize the data. Before you can do that, you will need to execute the following *Jupyter magic command*.

In [149…

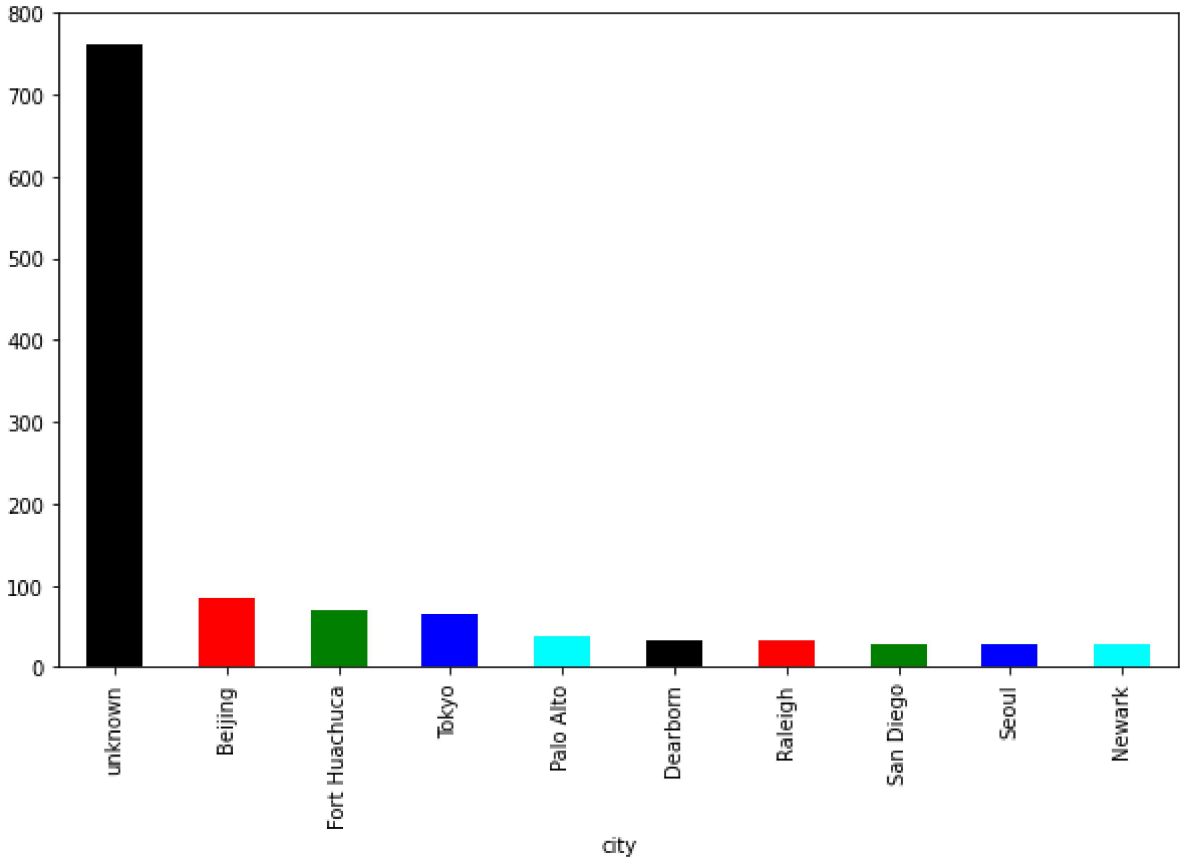
```
%matplotlib inline
```

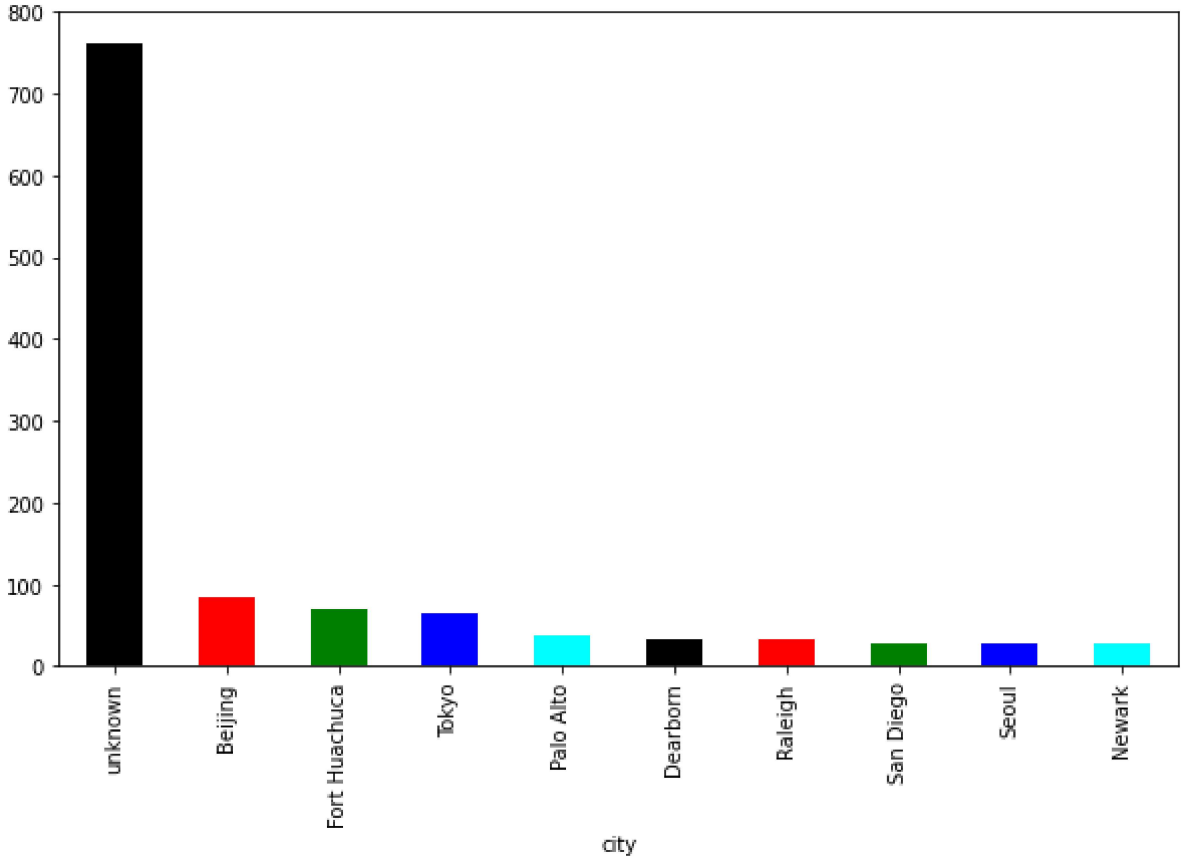
Let's concentrate on the most frequent cities. Create a bar chart showing the 10 most common cities among the website visitors.

In [150…

```
import matplotlib.pyplot as plt
fig,ax = plt.subplots(figsize = (10,6))
sorted_countwise.head(10).plot(kind = 'bar', y = 'count',ax = ax, legend = False,col
fig
```

Out[150…





Congratulations: You have completed the first task. Now let's look at the second task.

Visualizing coordinates

The second task from the marketing department is to find out which region in the world the most website visitors come from. You don't have the tools you need to answer this question

quantitatively. Therefore, it would be a good idea to choose a visual approach: to present the coordinates in a scatter plot and then identify the clusters visually.

For a scatter plot based on the `'latitude'` and `'longitude'` columns, these entries need to be stored as numbers. What data type do they currently have?

In [151…
```
df.dtypes
```

Out[151…
```
ip            object
user_agent    object
country       object
city          object
longitude     object
latitude      object
dtype: object
```

It's likely that `pandas` didn't recognize that these columns are numerical columns due to missing values. Now you need to force `pandas` to represent these columns as numerical columns.

In *Preparing Data with pandas (Chapter 1)* you used the `pandas.to_numeric()` function for the first time. Use it now to convert the `'longitude'` and `'latitude'` columns to numeric columns. Missing values should be represented as `NaN`. If you set the `errors` parameter in `pandas.to_numeric()` to `'coerce'`, anything that is not a number will be represented as `NaN`. Convert the `'longitude'` and `'latitude'` columns to numeric columns.

In [155…
```
df.loc[:, 'longitude'] = pd.to_numeric(df.loc[:, 'longitude'],errors='coerce')
df.loc[:, 'latitude'] = pd.to_numeric(df.loc[:, 'latitude'],errors='coerce')
df.dtypes
```

Out[155…
```
ip             object
user_agent     object
country        object
city           object
longitude     float64
latitude      float64
dtype: object
```

Now print the first 20 rows of `df` (all columns).

In [156…
```
df.head(20)
```

Out[156…

	ip	user_agent	country	city	longitude	latitude
0	216.216.56.64	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	United States	None	-97.8220	37.7510
1	189.109.167.60	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	Brazil	None	-43.2192	-22.8305
2	249.136.57.232	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	unknown	unknown	NaN	NaN
3	10.7.64.175	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	unknown	unknown	NaN	NaN
4	222.4.129.210	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	Japan	Tokyo	139.7514	35.6850

	ip	user_agent	country	city	longitude	latitude
5	32.195.5.171	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	United States	None	-97.8220	37.7510
6	181.95.62.227	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	Argentina	Córdoba	-64.1611	-31.2389
7	56.219.74.171	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	United States	Raleigh	-78.6253	35.7977
8	205.159.70.242	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	United States	New York	-73.9885	40.7317
9	211.94.3.180	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	China	None	113.7266	34.7725
10	107.197.44.42	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	United States	Chicago	-87.6368	41.8998
11	211.46.208.64	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	Republic of Korea	Sejong	127.2924	36.5930
12	62.1.126.114	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	Greece	None	23.7167	37.9667
13	157.171.197.86	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:5...	Sweden	Gothenburg	11.9667	57.7167
14	249.26.135.174	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	unknown	unknown	NaN	NaN
15	143.154.218.199	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	United States	Montgomery	-86.2539	32.4040
16	110.180.26.140	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59....	China	Xi'an	108.9286	34.2583
17	236.24.118.99	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59....	unknown	unknown	NaN	NaN
18	252.191.102.202	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	unknown	unknown	NaN	NaN
19	145.65.63.150	Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl...	Netherlands	Arnhem	5.9256	51.9798

There are now actually missing values in the two columns on the right. Is the data type in these columns a numeric data type?

In [157…
```
df.dtypes
```

Out[157…
```
ip            object
user_agent    object
country       object
city          object
longitude    float64
latitude     float64
dtype: object
```

There are missing values in the two columns on the right. Do the entries in these columns have a numeric data type?

In [158…
```
df.dtypes
```

Out[158…
```
ip             object
user_agent     object
country        object
city           object
longitude     float64
latitude      float64
dtype: object
```

Apparently so. That means that you can visualize the coordinates in a scatter plot.This is not necessarily the usual way to display geo-coordinates visually. `matplotlib` offers something called *Basemap* which provides differnet possibilities to add data to maps. This gallery shows what you can do with it. It's too much to cover in this course, so we'll limit ourselves to a simple scatter plot.

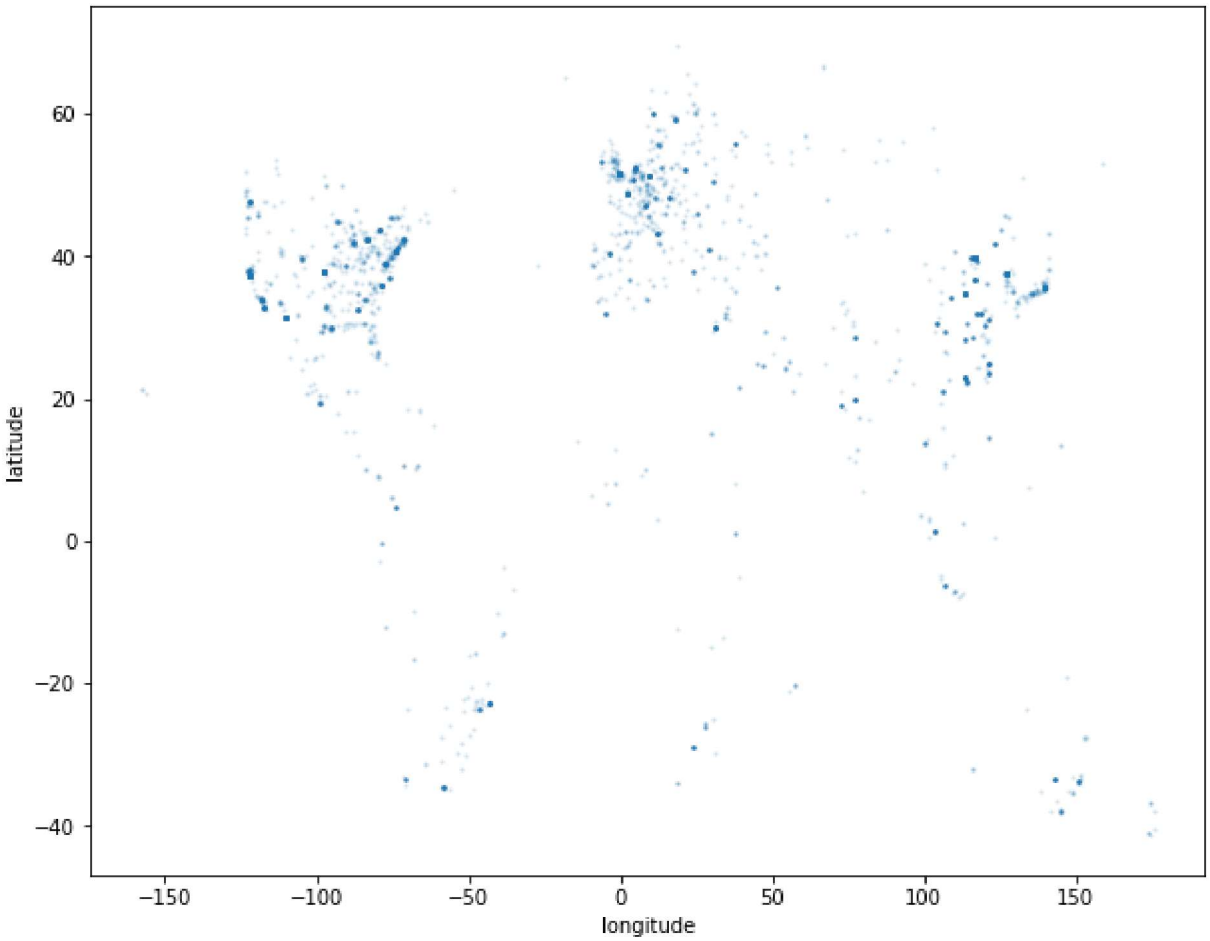
The longitude values are the east west coordinates for a point on the planet. 0 degrees longitude goes through Greenwich in the United Kingdom. Germany is to the east of that line, so it has a positive longitude value. The USA is to the west of Greenwich, so it has a negative longitude value.

The latitude values are the North-South coordinates for a point on the planet. The equator has a latitude of 0 degrees. Everything to the north of this, for example Europe, has a positive latitude value. Everything to the south of the equator has a negative latitude value.

In a scatter graph, the `'longitude'` column in `df` should form the x axis and the `'latitude'` column should be shown on the y axis. Draw a scatter graph in this way and use small, semi-transparent points. Set the `alpha` (transparency) parameter of `my_df.plot()` to 0.1 and the `s` parameter (size) to 2.

In [165…
```
fig,ax = plt.subplots(figsize  =(10,8))
df.plot(kind = 'scatter', x = 'longitude', y = 'latitude',ax =ax, alpha = 0.1,s = 2)
```

Out[165…
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a5df60fd0>
```



Even though there aren't any reference points, you can just about make out the continents. In the west, you can see the Americas. The USA and particularly the region around New York stand out in particular. In the middle you can see Northern Europe, in particular north-western Europe. There weren't many visitors from Africa. There are hardly any points in the middle of the scatter plot. In the east, you can see the densely populated areas in China, Korea and Japan. You can pass your observations on to the marketing department.

Congratulations: You have identified clusters in the coordinates using a scatter plot. You also learned how to access nested JSON data. The marketing department is very grateful that you were so helpful in this chapter and they hope you enjoy the rest of the course.

Remember:

- Access nested dictionaries with the keys from the highest level downwards:
  `dictionary['higher_key']['lower_key']['even_lower_key']`

Do you have any questions about this exercise? Look in the forum to see if they have already been discussed.

Found a mistake? Contact Support at support@stackfuel.com.
