In [ ]:
```python
import pandas as pd

# use read_json to convert json to dataframe
df = pd.read_json('data/simple.json')
```

In [ ]:
```python
# print the whole dataframe table
df
```

Out[ ]:

	id	name	math	physics	chemistry
0	A001	Tom	60	66	61
1	A002	James	89	76	51
2	A003	Jenny	79	90	78

In [ ]:
```python
# print just the first row
df.loc[0]
```

Out[ ]:
```
id           A001
name          Tom
math           60
physics        66
chemistry      61
Name: 0, dtype: object
```

In [ ]:
```python
# print first to second row
df.loc[[0, 1]]
```

Out[ ]:

	id	name	math	physics	chemistry
0	A001	Tom	60	66	61
1	A002	James	89	76	51

In [ ]:
```python
# print dataframe info. fields with numbers are defaulted to int64 instead of object
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   id         3 non-null      object
 1   name       3 non-null      object
 2   math       3 non-null      int64 
 3   physics    3 non-null      int64 
 4   chemistry  3 non-null      int64 
dtypes: int64(3), object(2)
memory usage: 248.0+ bytes
```

In [ ]:
```python
# read_json can also convert json in a file located in a url
URL = 'http://raw.githubusercontent.com/BindiChen/machine-learning/master/data-analy
df = pd.read_json(URL)
```

same results as using a local file

In [ ]:

```
df
```

Out[ ]:

	id	name	math	physics	chemistry
0	A001	Tom	60	66	61
1	A002	James	89	76	51
2	A003	Jenny	79	90	78

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   id         3 non-null      object
 1   name       3 non-null      object
 2   math       3 non-null      int64 
 3   physics    3 non-null      int64 
 4   chemistry  3 non-null      int64 
dtypes: int64(3), object(2)
memory usage: 248.0+ bytes
```

In [ ]:

```
# if we read a JSON with nested list it will be put into a single column
df = pd.read_json('data/nested_array.json')
df
```

Out[ ]:

	school_name	class	students
0	ABC primary school	Year 1	{'id': 'A001', 'name': 'Tom', 'math': 60, 'phy...
1	ABC primary school	Year 1	{'id': 'A002', 'name': 'James', 'math': 89, 'p...
2	ABC primary school	Year 1	{'id': 'A003', 'name': 'Jenny', 'math': 79, 'p...

to flatten the nested list, we can use json_normalize() function. in a way, when we specify the record_path, it's like treating that json field as an individual json file

In [ ]:

```
import json
# load data using Python JSON module
with open('data/nested_array.json','r') as f:
    data = json.loads(f.read())
# Flatten data
df_nested_list = pd.json_normalize(data, record_path =['students'])
```

In [ ]:

```
df_nested_list
```

Out[ ]:

	id	name	math	physics	chemistry
0	A001	Tom	60	66	61
1	A002	James	89	76	51
2	A003	Jenny	79	90	78

now to include other fields in a flattened nested list, we can use the meta parameter. here we are concatenating the school_name and class field in the flattened list

In [ ]:
```
# To include school_name and class
df_nested_list = pd.json_normalize(
    data,
    record_path =['students'],
    meta=['school_name', 'class']
)
```

In [ ]:
```
df_nested_list
```

Out[ ]:

	id	name	math	physics	chemistry	school_name	class
0	A001	Tom	60	66	61	ABC primary school	Year 1
1	A002	James	89	76	51	ABC primary school	Year 1
2	A003	Jenny	79	90	78	ABC primary school	Year 1

In [ ]:
```
# test.json has 2 fields 'school_name' and 'students' that has an array as their val
# read.json throws ValueError if not all arrays are of the same length
df = pd.read_json('data/test.json')
df
```

Out[ ]:

	school_name	class	students
0	ABC primary school	Year 1	{'id': 'A001', 'name': 'Tom', 'math': 60, 'phy...
1	DEF primary school	Year 1	{'id': 'A002', 'name': 'James', 'math': 89, 'p...
2	GHI primary school	Year 1	{'id': 'A003', 'name': 'Jenny', 'math': 79, 'p...

now we can flatten nested list in json file, what if there are nested list and dict in a json object?

we can again use the meta parameter in json_normalize but to use [] for example ['info', 'contacts', 'tel'] to contatenate a fields from a nested dict into our flattened list

In [ ]:
```
import json
# load data using Python JSON module
with open('data/nested_mix.json','r') as f:
    data = json.loads(f.read())

# Normalizing data
df = pd.json_normalize(
    data,
    record_path =['students'],
    meta=[
        'class',
        ['info', 'president'],
        ['info', 'contacts', 'tel']
    ]
)
df
```

Out[ ]:

	id	name	math	physics	chemistry	class	info.president	info.contacts.tel

	id	name	math	physics	chemistry	class	info.president	info.contacts.tel
0	A001	Tom	60	66	61	Year 1	John Kasich	123456789
1	A002	James	89	76	51	Year 1	John Kasich	123456789
2	A003	Jenny	79	90	78	Year 1	John Kasich	123456789

another scenario that we might run into is when we don't want to flatten the whole nested list but to **extract a single field** from a nested list

in that case we can use read_json with glom

In [ ]:

```
from glom import glom
df = pd.read_json('data/nested_deep.json')
df['students'].apply(lambda row: glom(row, 'grade.math'))
```

Out[ ]:

```
0    60
1    89
2    79
Name: students, dtype: int64
```